SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 1-18, January 1993

THE CONVERGENCE OF AN EXACT DESINGULARIZATION FOR VORTEX METHODS*

T. Y. HOU†, J. LOWENGRUB‡, AND M. J. SHELLEY§

Abstract. Expanding upon an observation of Hou [*Math. Comp.*, submitted], exact desingularizations are presented of the Euler equations in two and three dimensions for which the singularity within the Biot-Savart integrand is reduced by one order. The reformulated equations are then solved numerically using either the point vortex method or the vortex blob method. The increased smoothness of the Biot-Savart integrand allows us to prove convergence of these methods in the maximum norm. Our numerical experiments show that discretization of the reformulated equations display increased stability relative to discretizations of the original equations. The improvement in stability is manifested as a more slowly growing error in time.

Key words. vortex method, exact desingularization

AMS(MOS) subject classifications. primary 65M25; secondary 76C05

1. Introduction.

Expanding upon an observation of Hou [16], we present exact reformulations of the Euler equations in two and three dimensions for which the singularity within the Biot-Savart integrand is reduced by one order. The reformulated equations are then solved numerically using either the point vortex method or the vortex blob method. The increased smoothness of the Biot-Savart integrand allows us to prove convergence of these methods in the maximum norm. Our numerical experiments show that discretization of the reformulated equations display increased stability relative to discretizations of the original equations. The improvement in stability is manifested as a more slowly growing error in time.

In two dimensions, our reformulation of the Euler equations is based on the following simple observation: the velocity at the center of a radially symmetric distribution of vorticity is exactly zero. At each point in the fluid we modify the Biot-Savart integral. We subtract a specified vorticity distribution which is radially symmetric about the fluid point. This vorticity distribution is smooth, compactly supported, and its value at the center is equal to the fluid vorticity at that point. We then discretize the resulting equations using a point or blob vortex method. The discretized integrand will be one order less singular than the original point vortex discretization [25], provided that the vorticity is at least Lipschitz continuous. We refer to such a reformulation as an exact desingularization of the Biot-Savart integral. We also make similar observations and operations in three dimensions.

It is crucial to our results that the subtracted vorticity distribution be smooth and of compact support. In [16], Hou advocated subtracting and adding at each point in the fluid a constant vorticity distribution, with value equal to the fluid vorticity at that point. This constant vorticity was supported on a set that contains the fluid vorticity. A trapezoidal rule was then applied to a modified Biot-Savart integral, with the added

* Received by the editors June 18, 1990; accepted for publication October 2, 1991.

† Courant Institute, New York University, 251 Mercer Street, New York, New York 10012. The work of this author is partially supported by Air Force Office of Scientific Research grant AFOSR-90-0090.

‡ Department of Mathematics, Stanford University, Stanford, California 94305. Present address, Department of Mathematics, University of Minnesota, Minneapolis, Minnesota 55455. The work of this author is partially supported by the National Science Foundation and the Office of Naval Research.

§ Department of Mathematics, University of Chicago, 5734 University Avenue, Chicago, Illinois 60637. Present address, Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The work of this author is partially supported by a National Science Foundation Postdoctoral Research Fellowship.

vorticity handled through other means [7]. Again, the order of the singularity in the integrand was reduced. However, because the subtracted vorticity in Hou's original method does not vanish smoothly at the boundary of the computational support, the trapezoidal rule for a bounded domain gives, at best, second-order accuracy. This is in contrast to the higher-order accuracy that would be obtained if the vorticity vanished smoothly [11]. Actually, it is more appropriate to divide the computational domain into boxes and use a composite midpoint rule. Careless use of a trapezoidal rule may result in only first-order accuracy. In the method presented here, this disadvantage is circumvented because the subtracted vorticity is smooth and of compact support. Thus, the standard trapezoidal rule will provide second-order approximation for the point vortex method and high-order approximation for the vortex blob method.

The fact that our method is stable in maximum norm allows us to prove the convergence of a particular local regridding scheme. The local regridding method is given in [18] and we do not present it here. We simply remark that the existence of its proof of convergence now gives a theoretical basis for other similar regridding algorithms.

Although it may be tempting to make the subtracted vortex local, we show that the error constants grow as the support of the vortex diminishes. This is not surprising, because the method becomes the original point vortex method as the support of the subtracted vortex vanishes. This is also reflected in our numerical experiments: the larger the support of the subtracted vortex, the more slowly growing the error is in time. Thus, the size of the subtracted vortex should not be coupled to the mesh size. By our choice of reformulation, the discretized sums remain of convolution type, and the existence of fast summation algorithms for such sums is known [5]. Our numerical calculations were performed using direct summation on a Stardent GS2000 computer where the code was both vectorized and run in parallel on four processors.

Our computational domain must now include some marker particles that lie outside the support of the vorticity. This is due to the fact that we must accurately represent the subtracted vortices centered on the boundary of the support of the vorticity.

In addition, as our subtracted vortices are smooth, we are able to obtain an asymptotic error expansion for the point vortex approximation. In principle, the existence of such an expansion allows us to extrapolate to get higher-order accurate methods, and the increased stability of the basic new scheme enables these higher-order accurate methods to be more stable as well.

2. The equations of motion and vortex methods.

The Euler equations of two-dimensional, incompressible fluid flow in the vorticity-stream formulation are given by

$$\omega_t + u \cdot \nabla\omega = 0 \tag{1}$$

and

$$u(x, t) = \int_{\mathbf{R}^2} K(x-y)\omega(y, t)\, dy, \tag{2}$$

where $u(x, t) = (u_1(x, t), u_2(x, t))$, $x = (x_1, x_2)$ is the fluid velocity, $\omega(x, t)$ is the scalar vorticity, and

$$K(x) = \frac{1}{2\pi|x|^2}(-x_2, x_1) \tag{3}$$

is the Biot-Savart kernel. Equation (1) demonstrates that the vorticity in two dimensions is conserved along particle paths, while (2) (the Biot-Savart integral) gives the relation between the velocity and the vorticity due to the incompressibility constraint, and the definition of the vorticity.

We introduce the vortex or particle method by rewriting (1) and (2) in Lagrangian coordinates. $X(\alpha, t)$ is defined so that $X(\alpha, t)$ gives the position at time t of a particle that started at position α. That is,

$$\frac{dX(\alpha, t)}{dt} = u(X(\alpha, t), t), \qquad X(\alpha, 0) = \alpha. \tag{4}$$

Now by (1) we have

$$\omega(X(\alpha, t), t) = \omega(\alpha, 0) = \omega_0(\alpha), \tag{5}$$

since vorticity is conserved along particle paths. This yields the Lagrangian formulation:

$$\begin{aligned} \frac{dX(\alpha, t)}{dt} &= \int_{\mathbf{R}^2} K(X(\alpha, t) - X(\alpha', t))\omega_0(\alpha')\, d\alpha', \\ X(\alpha, 0) &= \alpha. \end{aligned} \tag{6}$$

The vortex method is concerned with the solution of this infinite-dimensional set of ordinary differential equations. Note that $K(x)$ is singular at the origin with the integrable singularity $1/|x|$.

The simplest and oldest vortex method consists of approximating the integral in (6) by the trapezoidal rule, omitting the singular self-interaction term, and solving the differential equations

$$\frac{d\tilde{X}_i(t)}{dt} = \sum_{j \neq i} K(\tilde{X}_i - \tilde{X}_j)\omega_j h^2, \qquad \tilde{X}_i(0) = \alpha_i = (i_1 h, i_2 h), \tag{7}$$

where $\omega_j = \omega_0(\alpha_j)$. This is the point vortex method (PVM) [25]. It has the following difficulty: as two point vortices approach one another, the velocity each induces on the other becomes unbounded. That is, there can be strong grid-scale interactions that lead to a rapid loss of accuracy in numerical calculations.

Nonetheless, this method has been proved convergent by Goodman, Hou, and Lowengrub [13]. A necessary condition for the stability of this scheme is that the method be accurate enough over a given time interval. The accuracy, however, depends on the grid size, the gradients of the solution, and the properties of the kernel K. In typical calculations though, this required accuracy can be lost in time rather quickly due to the development of large velocity gradients (which result in large Lagrangian grid distortions) and the singular nature of the kernel K (see [3], [13], and [23], for example).

To alleviate these difficulties, Chorin [8] introduced the idea of using a mollified kernel, K_δ, or equivalently, vortex blobs with a fixed shape, instead of point vortices to approximate the flow

$$\frac{d\tilde{X}_i^\delta(t)}{dt} = \sum_j K_\delta(\tilde{X}_i^\delta - \tilde{X}_j^\delta)\omega_j h^2, \qquad \tilde{X}_i^\delta(0) = \alpha_i, \tag{8}$$

where $K_\delta = K * f_\delta$, $f_\delta(x) = (1/\delta^2)f(x/\delta)$ where f is an mth order cutoff function satisfying

(i) $\displaystyle\int_{\mathbf{R}^2} f(x)\, dx = 1,$

(ii) $\displaystyle\int_{\mathbf{R}^2} x^\beta f(x)\, dx = 0$ for all multi-indices β such that $i \leqq |\beta| \leqq m - 1$.

This gives a computationally more stable method than the point vortex method if the smoothing size δ is larger than the grid size h. Convergence of the vortex blob method (VBM) has been well established. See [1]-[4], [9], [12], [15], and [24] as well as the review articles [6], [19], and [20] for a bibliography.

The grid distortion exhibited in the PVM is still present for the VBM, although its effects are not as catastrophic due to the mollification of K by K_δ. Consequently, both the VBM and the PVM produce relatively large errors for large time calculations [3], [13], [23]. The purpose of this paper is to introduce a new desingularization to maintain the overall accuracy for longer times with the same initial grid sizes.

3. Our method. We now present our method. It is based on the desingularization idea of Hou [16] and our observation that the velocity at the center of a radially symmetric vortex is zero.

Let $g(x)$ be a smooth function of compact support satisfying

$$g(x)=g(|x|)=\begin{cases}1, & |x|\leqq R_1,\\ 0, & |x|\geqq R_2.\end{cases} \tag{9}$$

We see that

$$\begin{aligned}\frac{dX_i}{dt}&=\int_{\mathbf{R}^2} K(X_i-X(\alpha))\omega_0(\alpha)\,d\alpha\\ &=\int_{\mathbf{R}^2} K(X_i-X(\alpha))(\omega_0(\alpha)-\omega_i g(X_i-X(\alpha)))\,d\alpha,\end{aligned} \tag{10}$$

since

$$\omega_i\int_{\mathbf{R}^2} K(X_i-X(\alpha))g(X_i-X(\alpha))\,d\alpha=\omega_i\int_{\mathbf{R}^2} K(x)g(x)\,dx=0, \tag{11}$$

as $X(\alpha)$ is an area-preserving transformation and $K(x)g(x)$ is an odd function in x. Again, this is just the statement that the velocity at the center of a radially symmetric vorticity distribution is zero. We refer to (10) as an exact desingularization of the Biot-Savart integral. The point vortex and vortex blob quadratures of (10) will be referred to as the desingularized point vortex and desingularized vortex blob methods (DPVM and DVBM).

In [16], Hou used $g(x)\equiv 1$, where the integrals in (10) and (11) are taken over the support of the vorticity (or some set containing it), assumed to be finite, and denoted by Ω. He considered the reformulated equations

$$\frac{dX_i}{dt}=\int_\Omega K(X_i-X(\alpha))(\omega_0(\alpha)-\omega_i)\,d\alpha+\omega_i\int_\Omega K(X_i-X(\alpha))\,d\alpha.$$

The second integral is the velocity induced at X_i by a patch of constant vorticity of strength ω_i and filling Ω. This integral is no longer zero, but can be computed explicitly over triangulations of the underlying mesh, or as a boundary integral [7]. However, note that the subtracted vorticity distribution does not vanish smoothly at $\partial\Omega$. Therefore, discretizations of the first integral must be based upon quadrature rules that take careful account of the boundary; misrepresentation of the boundary can lead to $O(h)$ errors. This problem is completely circumvented by using a subtracted vorticity distribution that is smooth and decaying (compact support). In addition, our method is simpler, as there is no correction term to calculate (the second integral above).

Here, we consider the point vortex and blob discretizations of (10), or

$$\tag{12} \begin{aligned} \frac{d\tilde{X}_i^\delta(t)}{dt} &= \sum_{\substack{j\neq i\\ jh\in\Delta_h}} K_\delta(\tilde{X}_i^\delta-\tilde{X}_j^\delta)(\omega_j-\omega_i g(\tilde{X}_i^\delta-\tilde{X}_j^\delta))h^2 \equiv \tilde{u}_i^\delta,\\ \tilde{X}_i^\delta(0) &= \alpha_i. \end{aligned}$$

To ensure the consistency of our scheme, our discretization must cover a set $\tilde{\Omega}$, which contains Ω, with the property that

$$\tag{13} \min \operatorname{dist}(\partial\tilde{\Omega}, \partial\Omega) \geqq R_2.$$

This condition is necessary to ensure that the discretization of (11) is zero to $O(h^2)$ for those points $X(\alpha_i, t)$ such that $\operatorname{dist}(X(\alpha_i, t), \partial\Omega) \leqq R_2$. That is, we carry marker particles in the region $\tilde{\Omega}-\Omega$. These particles carry no vorticity but will effect the velocity calculation at points $X_i \in \Omega$ such that $\operatorname{dist}(X_i, \partial\Omega) \leqq R_2$.

We now present our convergence theorems.

THEOREM 1 (convergence of the DPVM). *Assume that* $\omega_0, g \in C_0^2(\mathbf{R}^2)$ *and that* g *satisfies* (9) *and* $\tilde{\Omega}$ *satisfies* (13). *Then, for* $0\leqq t\leqq T$, *there exists* $h_0(\omega_0, T, g)$ *such that*

$$\max_i |X_i-\tilde{X}_i| \leqq C(T)h^2 \quad \text{and} \quad \max_i |u_i-\tilde{u}_i| \leqq C(T)h^2,$$

for all $0<h\leqq h_0$, *where* $X_i = X(\alpha_i, t)$ *and* $u_i = u(X_i, t)$ *are the exact solutions of the Euler equations and* $\tilde{X}_i$ *and* $\tilde{u}_i$ *are the solutions of* (12) *with* $\delta = 0$.

THEOREM 2 (convergence of DBVM). *Assume that* $f(x)$ *is an mth order cutoff function,* $g, f, \omega_0 \in C_0^r(\mathbf{R}^2)$, g *satisfies* (9), *and* $\tilde{\Omega}$ *satisfies* (13). *We assume that* $\delta = h^q$ *with* q *such that* mq *and* $(1-q)r+2q$ *are both* $\geqq 1+2\alpha_0$ *for some* $\alpha_0>0$. *This is so that the method is more than first-order accurate. Then, there exists* $h_0(\omega_0, f, g, T)$ *for* $0\leqq t\leqq T$ *such that*

$$\max_i |X_i-\tilde{X}_i^\delta| \leqq C(T)\left(\delta^m + \left(\frac{h}{\delta}\right)^r \delta^2\right)$$

and

$$\max_i |u_i-\tilde{u}_i^\delta| \leqq C(T)\left(\delta^m + \left(\frac{h}{\delta}\right)^r \delta^2\right)$$

for all $h\leqq h_0$, *where* $\tilde{X}_i^\delta$ *and* $\tilde{u}_i^\delta$ *are the solutions of* (12).

Remark 1. (i) Thanks to a smooth g, we can obtain an asymptotic error expansion in the case $\delta = 0$, in even integer powers of h.

(ii) We will prove only Theorem 1. Theorem 2 is proved using the bounds of Theorem 1 and follows directly from the standard analyses of vortex methods (see [1]-[4], [9], [12], [15], and [24], for example).

Proof of Theorem 1. As usual, we divide the proof into two parts: consistency and stability. We begin with consistency.

CONSISTENCY LEMMA. *Suppose that* $g, \omega_0 \in C_0^\infty(\mathbf{R}^2)$, *and* $\tilde{\Omega}$ *satisfies* (13) *with* Δ_h *as its Lagrangian discretization. Then there exists* $h_0(T, g, \omega_0)$ *such that for all* $0<h\leqq h_0$

$$\tag{14} \begin{aligned} \rho_i &\equiv \int_{\mathbf{R}^2} K(X_i - X(\alpha))\omega_0(\alpha)\,d\alpha - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(X_i-X_j)(\omega_j-\omega_i g(X_i-X_j))h^2\\ &= C_2(X_i, t, g)h^2 + C_4(X_i, t, g)h^4 + \cdots + O(h^N) \end{aligned}$$

for any N. The constants $C_{2n}(X_i, t, g)$ $(n \geqq 1)$ are smooth functions of X_i. Furthermore, if g, $\omega_0 \in C_0^2(\mathbf{R}^2)$ then the expansion stops at the fourth order and we only obtain $\|\rho\|_{l^\infty} \leqq C(T)h^2$.

Proof of consistency. We rewrite ρ_i as follows:

$$\begin{aligned}\rho_i &= \int_{\mathbf{R}^2} K(X_i - X(\alpha))\omega_0(\alpha)\,d\alpha - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(X_i - X_j)(\omega_j - \omega_i g(X_i - X_j))h^2\\ &= \int_{\mathbf{R}^2} K(X_i - X(\alpha))(\omega_0(\alpha) - \omega_i g(X_i - X(\alpha)))\,d\alpha\\ &\quad - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(X_i - X_j)(\omega_j - \omega_i g(X_i - X_j))h^2,\end{aligned}$$

since the velocity at the center of a radially symmetric vortex patch is zero.

Using assumption (13) and the fact that g is smooth with compact support, we can apply the consistency argument of Goodman, Hou, and Lowengrub [13]. We need only to replace their Lemma 2 by the following lemma.

LEMMA 2′. *For any N, we have*

$$\begin{aligned}&K(X(0) - X(\alpha))(\omega_0(\alpha) - \omega_0(0)g(X(0) - X(\alpha)))\\ &\quad = l_0(\alpha) + l_1(\alpha) + l_2(\alpha) + \cdots + O(|\alpha|^N),\end{aligned}$$

where l_n is homogeneous of degree n in α. Then, using l_n instead of m_n in the proof of Lemma 1 in [13], *we get the result*

$$\rho_i = C_2h^2 + C_4h^4 + \cdots + O(h^N)$$

for any N. Furthermore, the coefficients are rapidly decaying, as before [13], *and thus we obtain the desired result.*

The proof of Lemma 2′ is accomplished by expanding $X(\alpha)$, $K(X(0) - X(\alpha))$, and $g(X(0) - X(\alpha))$ in power series in α and then multiplying them appropriately. The presence of the term $\omega_0(0)g(X(0) - X(\alpha))$ eliminates the most singular term that appears in Lemma 2 of [13]. This completes the proof of the Consistency Lemma.

We now turn to the Stability Lemma.

STABILITY LEMMA. *If*

$$\|X(t) - \tilde{X}(t)\|_{l^\infty} \leqq h^{1+s}$$

for $t \leqq T^*$ and any $s > 0$, then there exists $E(T)$ such that

$$\begin{aligned}&\max_i \left| \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(X_i - X_j)(\omega_j - \omega_i g(X_i - X_j))h^2 - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(\tilde{X}_i - \tilde{X}_j)(\omega_j - \omega_i g(\tilde{X}_i - \tilde{X}_j))h^2 \right|\\ &\quad \leqq E(T)\|X - \tilde{X}\|_{l^\infty}.\end{aligned} \tag{15}$$

Proof of stability. Define

$$F_{ij}(y) = K(y)(\omega_j - \omega_i g(y)) \tag{16}$$

and $e_i = X_i - \tilde{X}_i$. Furthermore, we define

$$\sigma_i = \sum_{\substack{jh\in\Delta_h\\ j\neq i}} F_{ij}(X_i - X_j)h^2 - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} F_{ij}(\tilde{X}_i - \tilde{X}_j)h^2. \tag{17}$$

By the mean value theorem, we have

$$\sigma_i = \sum_{\substack{jh\in\Delta_h\\ j\neq i}} \nabla F_{ij}(X_i - X_j + X_{ijk}) \cdot (e_i - e_j)h^2, \tag{18}$$

where $X_{ijk} = \theta_k(e_i - e_j)$ for some $|\theta_k| \leqq 1$ and $k = 1, 2$ denotes the component of ∇F. The mean value theorem is applicable here because F_{ij} is nonsingular in the region considered:

$$\begin{aligned} |X_i - X_j + X_{ijk}| &\geqq |X_i - X_j| - |X_{ijk}| \\ &\geqq |X_i - X_j| - 2h^{1+s} \\ &\geqq C(T)|\alpha_i - \alpha_j| - 2h^{1+s} \\ &\geqq \frac{C(T)}{2}|\alpha_i - \alpha_j| \end{aligned} \tag{19}$$

for $t \leqq T^*$ and h small enough, where we have used $|X(\alpha_i) - X(\alpha_j)| \geqq C(T)|\alpha_i - \alpha_j|$ due to the smoothness of the inverse flowmap X^{-1}.

Now we examine ∇F_{ij}. Differentiating both sides of (16), we get

$$\nabla F_{ij}(y) = \nabla K(y)(\omega_j - \omega_i g(y)) - K(y)\omega_i \nabla g. \tag{20}$$

Since $g(y)$ is smooth, we have by the mean value theorem $g(y) = g(0) + \nabla g(\xi) y$ for some $|\xi| \leqq |y|$. Recall that $g(0) = 1$. Thus we obtain

$$\nabla F_{ij}(y) = \nabla K(y)(\omega_j - \omega_i) - (\omega_i \nabla g(\xi)) y \nabla K(y) - K(y)\omega_i \nabla g. \tag{21}$$

As a consequence, we obtain

$$|\nabla F_{ij}(X_i - X_j + X_{ijk})| \leqq \frac{D(T)}{|\alpha_i - \alpha_j|}, \tag{22}$$

where we have used (19) and the boundedness of ∇g. Then it follows from (18) and (22) that

$$|\sigma_i| \leqq D(T)\|e\|_{l^\infty} \sum_{\substack{jh \in \Delta_h \\ j \neq i}} \frac{h^2}{|\alpha_i - \alpha_j|} \leqq E(T)\|e\|_{l^\infty}, \tag{23}$$

which proves the Stability Lemma.

We are now ready to complete the proof of Theorem 1. Define

$$e_i(t) = X_i(t) - \tilde{X}_i(t)$$

and

$$T^* = \inf\{t \mid 0 \leqq t \leqq T, \|e(t)\|_{l^\infty} \leqq h^{1+s}\}$$

for some $0 < s < 1$. We have

$$\frac{de_i}{dt}(t) = \sigma_i(t) + \rho_i(t).$$

It follows from the Consistency and Stability Lemmas that for $t \leqq T^*$,

$$\frac{d}{dt}\|e\|_{l^\infty} \leqq E(T)\|e\|_{l^\infty} + C(T)h^2. \tag{24}$$

Gronwall's inequality then implies that

$$\|e\|_{l^\infty} \leqq H(T)h^2 \quad \text{for } t \leqq T^*. \tag{25}$$

But $H(T)$ is independent of T^*. Thus, for h small enough, we have $\|e\|_{l^\infty} \leqq h^{1+s}/2$. Hence we conclude that $T^* = T$, which completes the proof of Theorem 1.

Remark 2. (i) Our parameters R_1, R_2 allow for the fine tuning of the error. Clearly, as the difference $R_2 - R_1$ increases, our stability constants will decrease. Increasing R_1 and fixing the distance $R_2 - R_1$ will also result in a decrease of the stability constants.

(ii) Decreasing R_2 or $R_2 - R_1$ will increase the stability constant because ∇g becomes larger; see (21).

(iii) As a result of our asymptotic error expansion, we can extrapolate, using different grid sizes, to obtain higher-order accurate methods. In this way, we do not need to use explicit values of the vorticity gradients, we simply need to know that they are smooth.

4. Numerical results. In this section, we present some results of numerical experiments in which both the DPVM and the DVBM are implemented to simulate the two-dimensional Euler equations. These results are only intended to be illustrative of the advantages of the increased stability for vortex methods based on exact desingularization. It is clear that there are many ways in which the method could be further improved and fine tuned.

On a simple test problem, the convergence rates predicted by the convergence theorems are demonstrated. It is shown that increased stability accompanies wider support of the cutoff function $g(|x|)$, as indicated in Remark 2(i). Furthermore, these methods are compared with both the classical point vortex and vortex blob methods. The results indicate that the DPVM and the DVBM show increased stability relative to these methods, and consequently remain more accurate over longer times.

As our test case, we consider the motion of a smooth, radially symmetric distribution of vorticity. Such distributions are steady solutions to the Euler equations, in which fluid particles move with constant speed on circles about the origin. In particular, we use

$$\omega(x) = \begin{cases} (1-|x|^2)^7 & \text{if } |x| \leqq 1, \\ 0 & \text{if } |x| > 1. \end{cases}$$

The velocity field is then given by

$$u(x) = f(|x|)\,\frac{(-x_2, x_1)}{|x|}$$

with speed

$$f(r) = \begin{cases} (1-(1-r^2)^8)/(16r) & \text{if } r \leqq 1, \\ 1/(16r) & \text{if } r > 1. \end{cases}$$

The vorticity itself lies in $C^6(R^2)$. This particular test case has been used by many others; see [23], for example. It is a nontrivial test case for a Lagrangian method, as there is a good deal of grid distortion and stretching. As an illustration, consider the motion of a material curve that at $t=0$ is the line segment connecting $(0,0)$ and $(1,0)$. By $t=20$ this material curve has increased its length eightfold. This particular material curve is also a coordinate line in our calculations.

For our methods, many choices must be made. First, a cutoff function $g(r)$ must be chosen, which may itself involve further parameters (say R_1 and R_2 in (9)). Furthermore, a discretization of the exactly desingularized Biot-Savart integral (10) must be specified. We have discussed the point vortex and vortex blob approximations, though perhaps there are others worth entertaining. For definiteness, we consider here the cutoff function

$$g(r) = \begin{cases} 1 & \text{if } r \leqq R_1, \\ (1 - s(r)^8)^7 & \text{if } R_1 < r < R_2, \\ 0 & \text{if } r \geqq R_2, \end{cases}$$

where $s(r)^2 = (r^2 - R_1^2)/(R_1^2 - R_2^2)$. Note that g also lies in $C^6(R^2)$, and generally should be chosen to be at least in the same continuity class as the vorticity. Figure 1 shows g with $R_1 = \frac{1}{2}$ and $R_2 = 1$.

Using the DPVM, with the above choices of R_1 and R_2, Fig. 2 shows the maximum error in velocity, $\|e(t)\|_\infty$, for $h = 0.2$ (medium dash), 0.1 (short dash), and 0.05 (solid), on $0 \leqq t \leqq 20$. The time integration method is a fourth-order, Adams-Moulton predictor-corrector [11]. The timestep was chosen small enough to remove time discretization errors from Fig. 2. Initially, the computational points were placed on a square grid over the vorticity. Here the set covering the vorticity was taken large enough to satisfy condition (13). That the error is of second-order is evident in Fig. 3, which displays $\log_2(\|e(2h, t)\|_\infty / \|e(h, t)\|_\infty)$ for $h = 0.05$. This function should be approximately two for a second-order method.

Now we consider the effect of varying both R_1 and R_2 upon the DPVM. Let $R_1 = \alpha/2$ and $R_2 = \alpha$. Then as $\alpha \downarrow 0$, we recover the classical point vortex approximation. For $h = 0.1$, Fig. 4 shows the error as α is varied as $\alpha = 1$ (solid), $\frac{1}{2}$ (short dash), $\frac{1}{4}$ (medium dash), and 0 (long dash, the classical point vortex approximation). We note that the initial error in velocity (the discretization error) for all four cases is approximately $2.0 \cdot 10^{-3}$. As indicated by our convergence theorems, α is chosen for stability

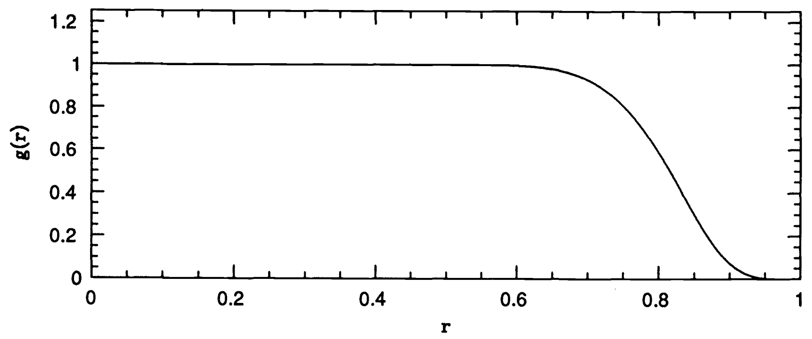


FIG. 1. *Graph of the patch function g with the parameters $R_1 = \frac{1}{2}$ and $R_2 = 1$.*

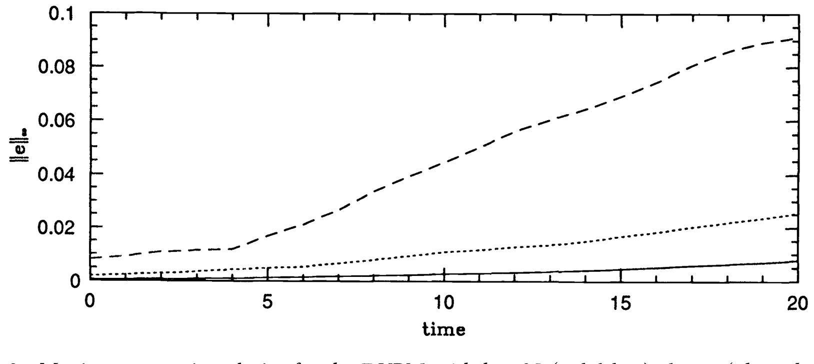


FIG. 2. *Maximum error in velocity for the* DVPM *with $h = .05$ (solid line), $h = .1$ (short dash), $h = .2$ (medium dash), and g of Fig. 1.*

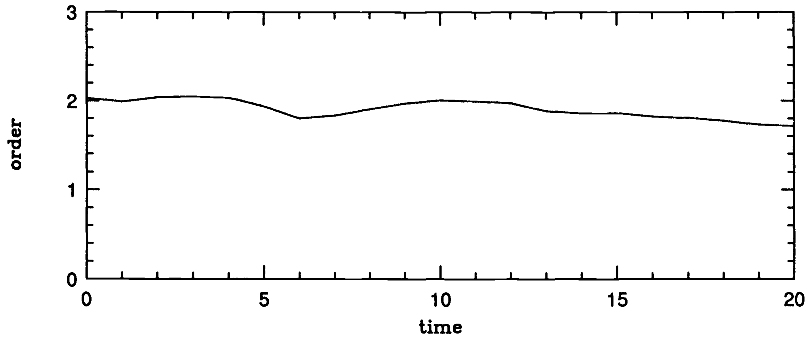


FIG. 3. *Order of convergence in maximum norm for velocity error for the* DVPM *with $h = .1$, $h = .05$, and the g of Fig.* 1.

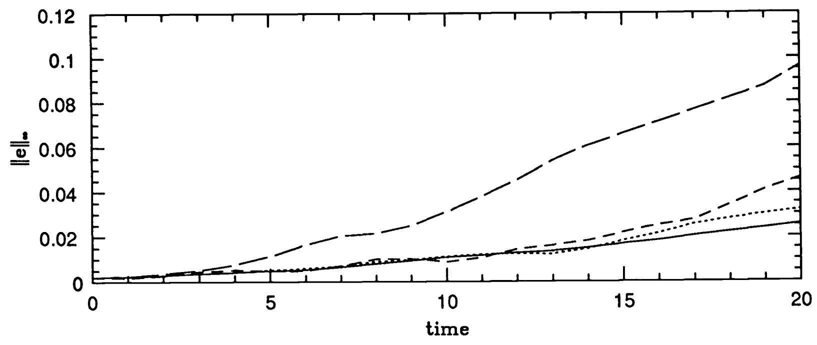


FIG. 4. *Comparison of the velocity error for the* DVPM *as the patch function g changes. We let $R_1 = \alpha/2$ and $R_2 = \alpha$. Then the curves are the error for $\alpha = 1$ (solid line), $\alpha = \frac{1}{2}$ (short dash), $\alpha = \frac{1}{4}$ (medium dash), $\alpha = 0$ (long dash), and $h = .1$.*

considerations, rather than for accuracy. Note from Fig. 4 that as α becomes larger and the support of the cutoff function grows, the growth of error decreases in time.

It is informative to decompose this error into that part associated with consistency and that with stability. As is usual, the error associated with consistency is given by

$$\int_{\mathbf{R}^2} K(X_i - X(\alpha))\omega_0(\alpha)\, d\alpha - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(X_i - X_j)(\omega_j - \omega_i g(X_i - X_j))h^2,$$

and that for stability by

$$\sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(X_i - X_j)(\omega_j - \omega_i g(X_i - X_j))h^2 - \sum_{\substack{jh\in\Delta_h\\ j\neq i}} K(\tilde{X}_i - \tilde{X}_j)(\omega_j - \omega_i g(\tilde{X}_i - \tilde{X}_j))h^2,$$

where $X_i(t)$ and $\tilde{X}_i$ are the exact and approximate solutions, respectively. Figure 5 shows these two errors (in maximum norm) for the classical PVM (dash) and for the desingularized PVM (solid). It is clear that the dominant error in both methods is that associated with stability, and that the stability control associated with the use of exact desingularization leads to much decreased error growth in time. An additional benefit from using exact desingularization here is a slower growth of the consistency error as the mesh becomes distorted and stretched.

We reach similar conclusions when comparing the DVBM with the classical VBM. Figure 6 shows the maximum error in velocity, with $h = 0.1$, for the DVBM (solid)

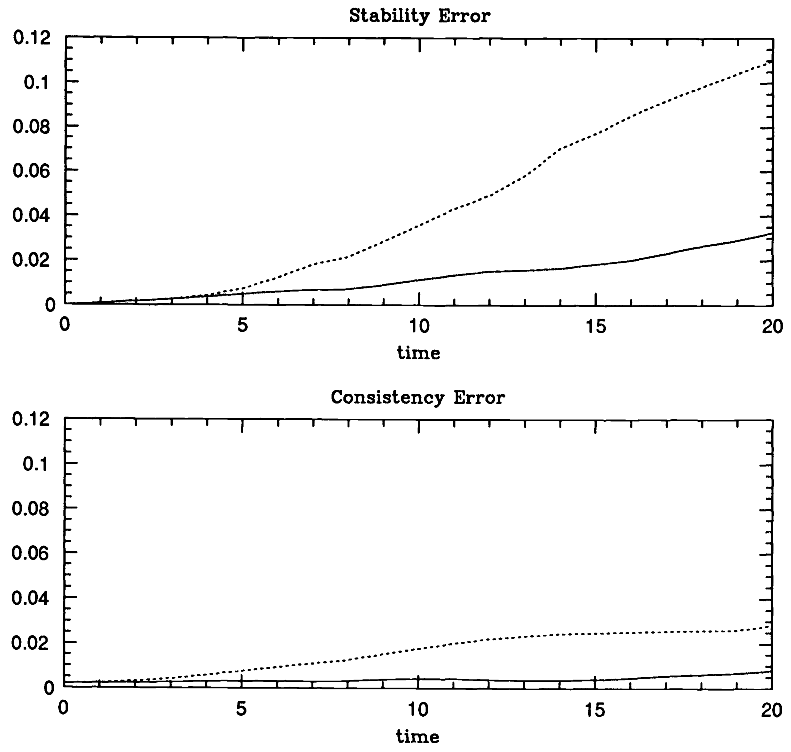


FIG. 5. (a) *Stability error for the* DVPM (*solid line*) *with g of Fig.* 1 *vs.* PVM (*dashed line*) *for* $h = .1$. (b) *Consistency error for the* DVPM *with g of Fig.* 1 (*solid line*) *vs.* PVM (*dashed line*) *for* $h = .1$.

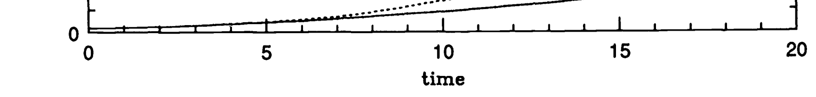


FIG. 6. *Comparison of the maximum error in velocity for the* DVBM *with g of Fig.* 1 (*solid line*) *vs.* VBM *with* $h = .1$, $\delta = h^{.95}$, *and a fourth-order cutoff function* (*Beale and Majda* [3]).

and the classical VBM (dashed). Here we have used a fourth-order kernel (Beale and Majda [3]), with small blob size $\delta = h^{0.95}$. Note that the vertical scale in Fig. 6 is one-half that in Figs. 4 and 5. For this choice of δ, Theorem 2 gives an order of accuracy of $O(h^{2.2})$, which is only slightly higher than that of the PVM. However, we expect to have a further improvement in the stability of the scheme. This expectation is justified by Fig. 6. Also note that while the VBM was initially more accurate than the DPVM (Fig. 4) by $t = 20$ this situation has reversed, and the DPVM has become more accurate. Figure 7 shows the associated stability and consistency errors. Again, the method based

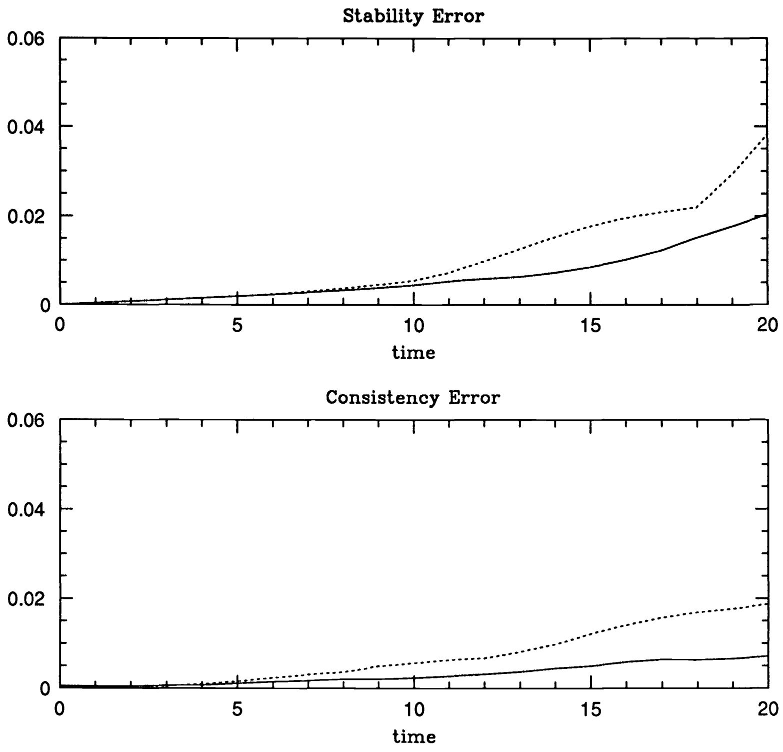


FIG. 7. (a) *Stability error for* DVBM *with g of Fig.* 1 (*solid line*) *vs.* VBM (*dashed line*) *with* $h = .1$. (b) *Consistency error for* DVBM *with g of Fig.* 1 (*solid line*) *vs.* VBM (*dashed line*).

upon exact desingularization shows both decreased stability and consistency error relative to the standard methods.

Finally, we present some preliminary calculations of the interaction of two like-signed vortices. Again, these calculations illustrate the computational advantages of exact desingularization, but they also demonstrate that our methods offer no panacea to a fundamental source of error in Lagrangian methods, which is grid distortion and stretching. We consider the initial vorticity distribution

$$\omega(x, y) = (\max(0, (.25 - (|x| - .5)^2 - y^2)))^7/.25^7.$$

Contours of this vorticity distribution are shown in Fig. 8. Evolution from this initial condition was calculated by Nordmark [22], who showed that the accuracy of vortex methods could be much improved through periodic rezoning of the particle positions. We do not present a comparison with his work here, but only seek a more interesting example with which to study our methods.

The initial particle positions are on a square grid covering the vorticity distribution. To visualize the motion of the vortices, we have placed marker particles on the four contours of the vorticity shown in Fig. 8. Initial point positions are shown on the left set of contours. The inner two contours each have 50 points, while the outer two have 200 points. The outermost contours coincide with the boundary of the support of the vorticity. The velocity of a marker particle with position $Y(t)$ is calculated by

$$\frac{dY(t)}{dt} = \sum_{jh \in \Delta_h} K_\delta(Y - \tilde{X}_j^\delta)(\omega_j - \omega(Y(0))g(Y - \tilde{X}_j^\delta))h^2.$$

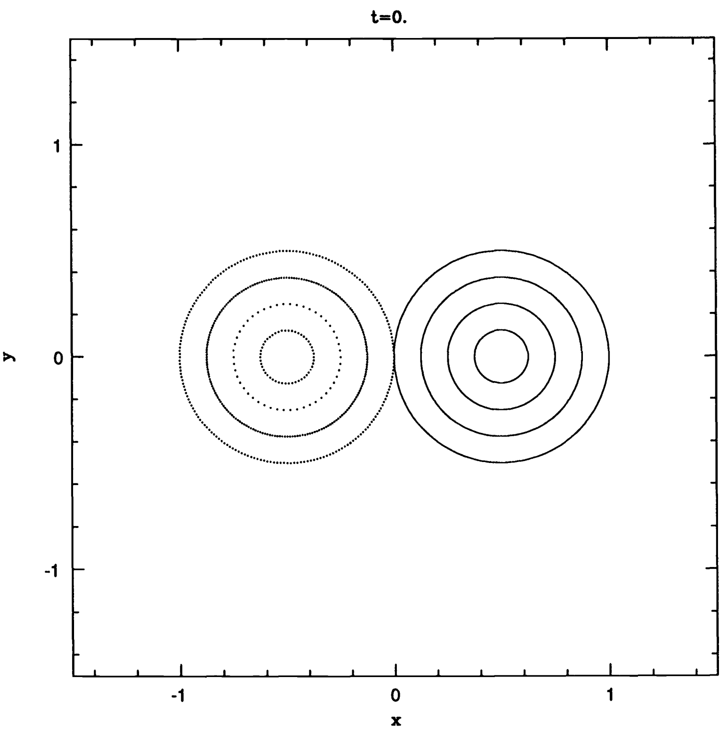


FIG. 8. *Initial position of marker particles for the two-patch problem.*

Figures 9(a) and 9(b) show the evolution of these marker particles at $t = 10.$ and $t = 20.$, using the DVBM with $h = .05$, $\delta = h^{.95}$, and $\alpha = 1$ for g, as given above. The lack of smoothness of the inner contours at $t = 20.$ is nonphysical, as Nordmark's careful calculations show. From these figures it is clear that vortex methods based upon exact desingularization would also benefit from a rezoning strategy. In Figs. 10(a) and 10(b) we show the same calculation, now using the classical VBM. The conclusions are self-evident.

We close this section with several remarks.

(i) The existence of fast summation techniques for a particle method is an important consideration. The first part of the sum (12),

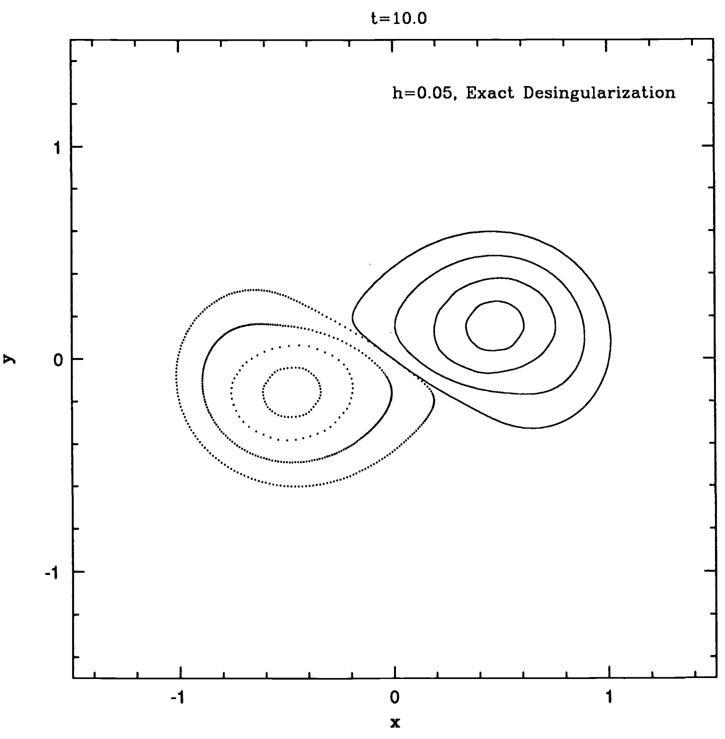
$$h^2 \sum_{j \neq i} K_\delta(\tilde{X}_i^\delta - \tilde{X}_j^\delta)\omega_j,$$

can be evaluated by methods already developed for Biot–Savart sums (see Greengard and Rokhlin [14]). Because of the nature of our cutoff function, the second part of the sum,

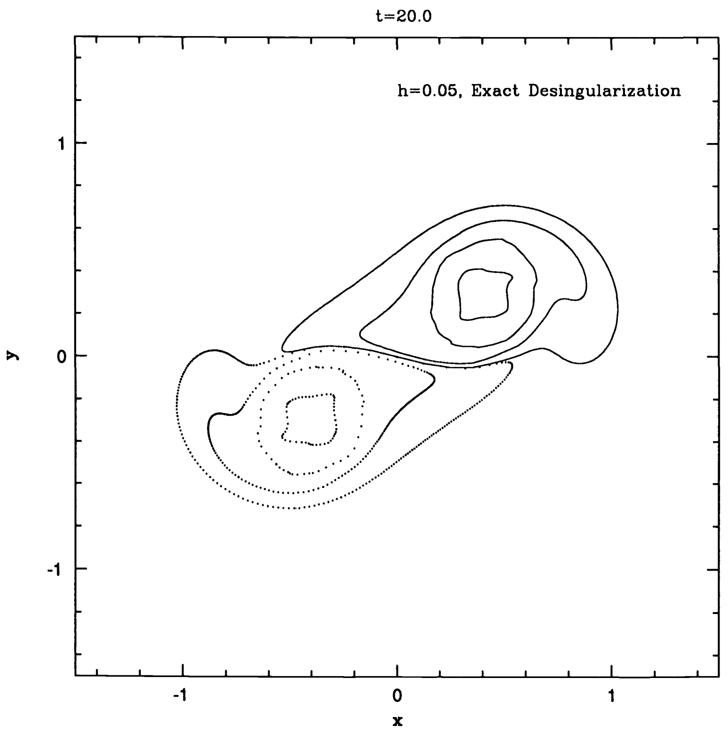
$$h^2\omega_i \sum_{j \neq i} K_\delta(\tilde{X}_i^\delta - \tilde{X}_j^\delta)g(\tilde{X}_i^\delta - \tilde{X}_j^\delta),$$

remains of convolution type. That is, $\hat{K} = K_\delta g$ is convolved with 1. General fast summation techniques have also been developed for such sums (see Brandt [5]).

(ii) We do not present results for three-dimensional calculations using the exact desingularization for the three-dimensional Euler equations to be discussed in § 5. We are currently implementing this method.

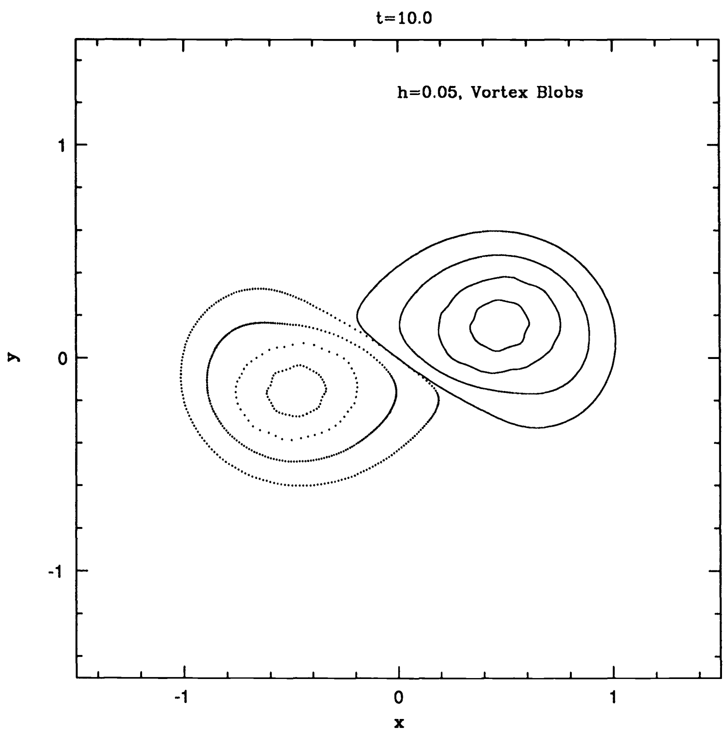


(a)

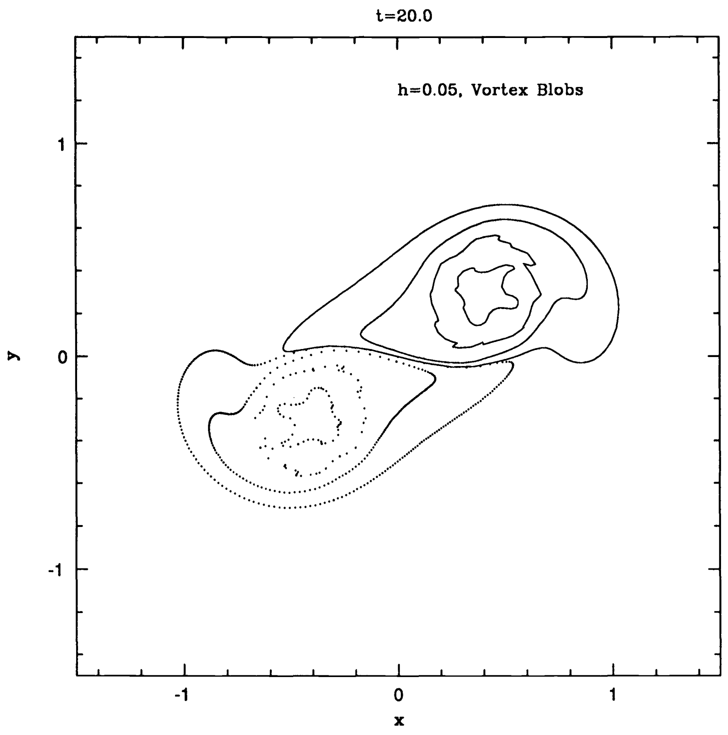


(b)

FIG. 9. (a) *Position of marker particles at* $t = 10$ *using the* DVBM *with* $h = .05$ *and* g *of Fig.* 1. (b) *Position of marker particles at* $t = 20$ *using the* DVBM *with* $h = .05$ *and* g *of Fig.* 1.



(a)



(b)

FIG. 10. (a) *Position of marker particles at* $t = 10$ *using the* VBM *with* $h = .05$. (b) *Position of marker particles at* $t = 20$ *using the* VBM *with* $h = .05$.

(iii) Merriman [21] has pointed out that further subtractions, which account for yet higher-order information of the vorticity, may not only increase stability, but also improve accuracy.

(iv) Evolving the marker particles in the region $\tilde{\Omega}-\Omega$ is costly. While the use of fast summation techniques will considerably ameliorate this cost, much can be done in optimizing the cutoff function g to reflect the vorticity distribution.

We now turn to the three-dimensional extension of our desingularization.

5. Generalization to the three-dimensional grid-free vortex method. The three-dimensional incompressible Euler equations in vorticity-stream function formulation are

$$w_t+(u\cdot\nabla)w=(w\cdot\nabla)u, \tag{26}$$

where

$$u(x,t)=\int_{\mathbf{R}^3}K(x-x')w(x',t)\,dx' \tag{27}$$

and

$$K(x)=\frac{-1}{4\pi|x|^3}\begin{pmatrix}0 & x_3 & -x_2\\ -x_3 & 0 & x_1\\ x_2 & -x_1 & 0\end{pmatrix}.$$

The particle formation is, using incompressibility,

$$\frac{dX}{dt}(\alpha,t)=u(X(\alpha,t),t)=\int_{\mathbf{R}^3}K(X(\alpha,t)-X(\alpha',t))w(X(\alpha',t),t)\,d\alpha' \tag{28}$$

and

$$\begin{aligned}\frac{dw}{dt}(X(\alpha,t))&=(w\cdot\nabla)u\\ &=w(X(\alpha,t),t)\cdot\int_{\mathbf{R}^3}\nabla K(X(\alpha,t)-X(\alpha',t))\cdot w(X(\alpha',t),t)\,d\alpha'.\end{aligned} \tag{29}$$

The reformulation idea presented in § 3 applies to the three-dimensional method as well.

As in § 3, we define a radial scalar function g, satisfying $g(|x|)=0$ for $|x|\geqq R_2$, $g(|x|)=1$ for $|x|\leqq R_1$, and smoothly varying in between. Then, it is clear that

$$\int_{\mathbf{R}^3}K(X(\alpha,t)-X(\alpha',t))w(X(\alpha,t),t)g(X(\alpha,t)-X(\alpha',t))\,d\alpha'=0, \tag{30}$$

since K is odd. Therefore,

$$\begin{aligned}&\int_{\mathbf{R}^3}K(X(\alpha)-X(\alpha'))w(X(\alpha'))\,d\alpha'\\ &\quad=\int_{\mathbf{R}^2}K(X(\alpha)-X(\alpha'))(w(X(\alpha'))-w(X(\alpha))g(X(\alpha)-X(\alpha')))\,d\alpha',\end{aligned} \tag{31}$$

which reduces the singularity of K by one order.

The particle equations (28), (29) can be desingularized as follows. Consider the integral

$$\begin{aligned}&\int_{\mathbf{R}^3}\nabla K(X(\alpha)-X(\alpha'))w(X(\alpha))g(X(\alpha)-X(\alpha'))\,d\alpha'\\ &\quad=\int_{\mathbf{R}^3}\nabla K(x-x')w(x)g(x-x')\,dx'.\end{aligned} \tag{32}$$

Without loss of generality, we consider the case $x=0$. Then (32) reduces to

$$\int_{\mathbf{R}^3} \nabla K(-x')w(0)g(-x')\,dx'. \tag{33}$$

The integrand is a matrix whose elements have the following form:

$$\frac{\pm 3x_i'x_j'}{|x'|^5}\gamma_k g(-x')+\frac{\kappa_{ij}\gamma_k g(-x')}{|x'|^5}[3x_i'^2-(x_1'^2+x_2'^2+x_3'^2)], \tag{34}$$

where $\kappa_{ij}=\pm 1$ if $i\neq j$, $\kappa_{ii}=0$, and $w(0)=(\gamma_1,\gamma_2,\gamma_3)$. Thus, since g is a radial function, the integral of the first term in (34) vanishes due to oddness in each coordinate axis. The integral of the second term vanishes due to exact cancellation of integrals.

Therefore, the reformulation of (28), (29) can be stated as:

$$\frac{dX}{dt}(\alpha,t)=\int K(X(\alpha)-X(\alpha'))(w(X(\alpha'))-w(X(\alpha))g(X(\alpha)-X(\alpha')))\,d\alpha', \tag{35}$$

$$\frac{dw}{dt}(\alpha,t)=\int \nabla K(X(\alpha)-X(\alpha'))(w(X(\alpha'))-w(X(\alpha))g(X(\alpha)-X(\alpha')))\,d\alpha', \tag{36}$$

with $X(\alpha,0)=\alpha$ and $w(\alpha,0)=w_0(\alpha)$. The numerical method is then given by:

$$\frac{d\tilde{X}_i}{dt}=\sum_{\substack{jh\in\Delta_h\\ j\neq i}} K_\delta(\tilde{X}_i-\tilde{X}_j)(\tilde{w}_j-\tilde{w}_i g(\tilde{X}_i-\tilde{X}_j))h^3, \tag{37}$$

$$\frac{d\tilde{w}_i}{dt}=\sum_{\substack{jh\in\Delta_h\\ j\neq i}} \nabla K_\delta(\tilde{X}_i-\tilde{X}_j)(\tilde{w}_j-\tilde{w}_i g(\tilde{X}_i-\tilde{X}_j))h^3, \tag{38}$$

with $\tilde{X}_i(0)=\alpha_i$ and $\tilde{w}_i(0)=w_0(\alpha_i)$, and where $\delta\geqq 0$.

We remark that this method is less physical than the corresponding two-dimensional method. The reason for this is that our requirement of compact support for g is tantamount to assuming that vortex lines end. However, we hope that the reduction of the singularity of the summands in (37) and particularly (38) makes three-dimensional computations more feasible.

The convergence of the desingularized method (37), (38) can be obtained by either following Beale's proof for the VBM [2] or by following the proof of Cottet, Goodman, and Hou [10] for the PVM. In either case, the stability is easier, and in the case of the PVM, the factor $\log(1/h)$ [10] can be eliminated.

Acknowledgments. It is a pleasure to thank G. Baker, R. Caflisch, J. Keller, and especially B. Merriman for their interest and valuable discussions concerning this work. The authors also wish to thank C. Anderson (Reviewing Editor, SISC) and a reviewer for their useful comments.

REFERENCES

[1] C. Anderson and C. Greengard, *On vortex methods*, SIAM J. Numer. Anal., 22 (1985), pp. 413–440.

[2] J. T. Beale, *A convergent 3-D vortex method with grid free stretching*, Math. Comp., 46 (1986), pp. 401–424.

[3] J. T. Beale and A. Majda, *High order accurate vortex methods with explicit velocity kernels*, J. Comp. Phys., 58 (1985), pp. 188–208.

[4] J. T. Beale and A. Majda, *Vortex methods* II: *High order accuracy in* 2 *and* 3 *dimensions*, Math. Comp., 32 (1982), pp. 29–52.

[5] A. Brandt and A. A. Lebrecht, *Multilevel Matrix Multiplication and Fast Solution of Integral Equations*, preprint, 1989.

[6] R. E. Caflisch, *Mathematical analysis of vortex dynamics*, in Mathematical Analysis of Vortex Dynamics, Society for Industrial and Applied Mathematics, Phila., PA, 1988, pp. 1-24.

[7] T. Chacon and T. Y. Hou, *A Lagrangian finite element method for the 2-D Euler equations*, Comm. Pure Appl. Math., to appear.

[8] A. J. Chorin, *A numerical study of slightly viscous flow*, J. Fluid Mech., 57 (1973), pp. 785-796.

[9] G. H. Cottet, *Methodes Particulaires pour l'Equation d'Euler dans le Plan*, These 3eme Cycle, Univ. P. et M. Curie, Paris, France, 1987.

[10] G. H. Cottet, J. Goodman, and T. Y. Hou, *Convergence of the grid free point vortex method for the three-dimensional Euler equations*, SIAM J. Numer. Anal., 28 (1991), pp. 291-307.

[11] G. Dahlquist and A. Bjork, Numerical Methods, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[12] J. Goodman and T. Y. Hou, *New stability estimates for the 2-D vortex method*, Comm. Pure Appl. Math., submitted.

[13] J. Goodman, T. Y. Hou, and J. Lowengrub, *Convergence of the point vortex method for the 2-D Euler equations*, Comm. Pure Appl. Math., 43 (1990), pp. 415-430.

[14] L. Greengard and V. Rokhlin, *A fast algorithm for particle summations*, J. Comput. Phys., 73 (1987), pp. 325-348.

[15] O. Hald, *Convergence of vortex methods* II, SIAM J. Numer. Anal., 16 (1979), pp. 726-755.

[16] T. Y. Hou, *A new desingularization for vortex methods*, Math. Comp., submitted.

[17] T. Y. Hou and J. Lowengrub, *Convergence of a point vortex method for the 3-D Euler equations*, Comm. Pure Appl. Math., 43 (1990), pp. 965-982.

[18] T. Y. Hou, J. Lowengrub, and M. J. Shelley, *Exact desingularization and local regridding for vortex methods*, Vortex Dynamics and Methods, C. Anderson and C. Greengard, eds., Lectures in Applied Mathematics, American Mathematical Society, Providence, RI, 1991.

[19] A. Leonard, *Computing three-dimensional incompressible flows with vortex elements*, Ann. Rev. Fluid Mech., 17 (1985), pp. 289-335.

[20] A. Majda, *Vortex dynamics: Numerical analysis, scientific computing and mathematical theory*, in ICIAM '87: Proceedings of the First International Conference on Industrial and Applied Mathematics, J. McKenna and R. Temam, eds., Society for Industrial and Applied Mathematics, Phila., PA, 1988.

[21] B. Merriman, personal communication.

[22] H. O. Nordmark, *Higher order vortex methods with rezoning*, Dept. of Mathematics, Ph.D. thesis, University of California, Berkeley, CA, 1988.

[23] M. Pearlman, *On the accuracy of vortex methods*, J. Comput. Phys., 59 (1985), pp. 200-223.

[24] P. A. Raviart, *An analysis of particle methods*, Centro Internazionale Matematico Estivo Course, Como, Italy, 1983.

[25] L. Rosenhead, *The point vortex approximation of a vortex sheet*, Proc. Roy. Soc. London Ser. A, 134 (1932), pp. 170-192.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 19-43, January 1993

SOME NUMERICAL ASPECTS OF COMPUTING INERTIAL MANIFOLDS*

ROBERT D. RUSSELL†, DAVID M. SLOAN‡, AND MANFRED R. TRUMMER†

Abstract. Various aspects of the problem of computing inertial manifolds for dissipative partial differential equations are investigated. In particular, methods are proposed which generalize previous ones and improve upon some of their limitations. The basic ideas and techniques are exemplified mainly for the Kuramoto-Sivashinsky equation. This equation is chosen because it is fairly simple but the dynamics are sufficiently complicated, and it is a classical case for which a considerable amount of computational experience has been well documented.

Key words. dissipative PDE, inertial manifold, inertial form, spectral method, Galerkin method, differential-algebraic equations, reaction diffusion equation, Kuramoto-Sivashinsky equation

AMS(MOS) subject classifications. 65P05, 65N30, 35K55, 35B32, 34C35, 34C30

1. Introduction. Recently there has been considerable interest, especially in the dynamical systems community, in the computation of inertial manifolds for dissipative partial differential equations (PDEs). This process is carried out by first splitting the solution space H into two subspaces PH and QH, where P is an orthogonal projection on H with finite-dimensional range and $Q = I - P$. The projection components in PH and QH are denoted, respectively, by p and q, and these components are governed by a system of ordinary differential equations (ODEs). The q-variables are expressed in terms of the p-variables by some relation $q = \phi(p)$, and the dynamics of this reduced system of ODEs for the p-variables are then used to mimic the dynamics of the underlying PDE.

We consider an evolution equation of the form

$$u_t + Au + F(u) = 0, \tag{1}$$

where A is a selfadjoint positive operator with compact resolvent (the dissipative part) defined on a Hilbert space H, and $F(u)$ is the nonlinear part defined on the domain of A. The dissipative nature of the PDE imparts it with a number of important properties (e.g., see [CFNT]), one of which is that the solution map $u_0 \to u(t) = S(t)u_0$ satisfies the following: There exists a compact, convex set $Y \subset H$ such that, for any bounded set $F \subset H$ there exists $t_0 = t_0(F)$ with $S(t)u_0 \in Y$ for all $t \geqq t_0$, $u_0 \in F$. We are interested in studying the long-term dynamics of the PDE. Although these dynamics can be extremely complicated, we can show under suitable general conditions that the PDE has an *inertial manifold* Ω, which is a manifold in Y with the following properties [LS]:

(i) Ω is invariant, i.e., $S(t)\Omega \subset \Omega$ for all $t \geqq 0$;
(ii) Ω attracts all solutions of (1) exponentially; and
(iii) Ω is a finite-dimensional C^1 manifold.[1]

* Received by the editors October 22, 1990; accepted for publication (in revised form) December 5, 1991.

† Department of Mathematics and Statistics, Simon Fraser University, Burnaby, B.C., V5A 1S6, Canada. The research of the first author was supported by Natural Sciences and Engineering Research Council of Canada grant A8781. The research of the third author was supported by Natural Sciences and Engineering Research Council of Canada grant OGP0036901.

‡ 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, and the Carnegie Trust.

[1] The C^1 smoothness is a regularity result following from the definition.

It follows that all of the long-term dynamics take place on this inertial manifold. In particular, the global attractor [CFNT] is a subset of this manifold. While the rate of attraction to the global attractor may be slow, all solutions are attracted to the inertial manifold exponentially fast.

For most PDEs of the form (1) which have been considered to date, the linear operator A is sufficiently simple (the negative Laplacian $-\Delta$ or bi-Laplacian Δ^2 operator) that a spectral method can be straightforwardly used to approximate $u(t)$. In particular, suppose that $w_1, w_2, \dots$ are the orthogonal eigenfunctions of A with corresponding eigenvalues $0 < \lambda_1 \leqq \lambda_2 \leqq \cdots$, so that

$$Aw_j = \lambda_j w_j, \qquad j = 1, 2, \dots . \tag{2}$$

Define P and $Q := I - P$ to be the usual spectral projections with P projecting onto the subspace spanned by the first n eigenfunctions, i.e.,

$$Pu = p := \sum_{j=1}^{n} \alpha_j(t) w_j := \sum_{j=1}^{n} \frac{(u, w_j)}{(w_j, w_j)} w_j, \tag{3}$$

where (,) is the scalar product on H. Under suitable conditions, such as that the eigenvalues grow sufficiently fast to satisfy a *gap condition* [LS], it is possible to determine an (upper bound on) n such that Ω is the graph of a function ϕ satisfying $QS(t)u_0 = \phi(PS(t)u_0)$ for all $t \geqq 0$ whenever $Qu_0 = \phi(Pu_0)$. In this case, Ω is an n-dimensional manifold in H. Since P and Q commute with A, it follows that not only is the PDE (1) equivalent to the ODE system

$$\dot{p} + APp + PF(p+q) = 0, \tag{4a}$$

$$\dot{q} + AQq + QF(p+q) = 0, \tag{4b}$$

but its long-term dynamics (solution stability, hyperbolicity, bifurcations) are completely determined by the finite-dimensional ODE system

$$\dot{p} + APp + PF(p + \phi(p)) = 0 \tag{5}$$

with *no* error. This last system is called an *inertial form* for (1).

In practice, Q is approximated by truncating its corresponding spectral projection, which under suitable conditions can be theoretically justified. To facilitate the presentation, we assume (unless stated otherwise) that this has been done for m sufficiently large and simply denote

$$Qu = q := \sum_{j=n+1}^{m} \alpha_j(t) w_j := \sum_{j=n+1}^{m} \frac{(u, w_j)}{(w_j, w_j)} w_j \tag{6}$$

for Q and q in (4b). As a consequence of this truncation process and the subsequent numerical solution, $\phi(p)$ is replaced by an approximation, say $\phi_a(p)$, and the corresponding graph Ω_a defines an approximate inertial manifold (AIM). Henceforth in this paper we shall assume that the truncation (6) has been effected: we shall drop the subscript a on ϕ and Ω and assume that Ω denotes an AIM.

Determination of the long-term solution behaviour for the dissipative PDE (1) now reduces to computing solutions to (5), and this requires that an approximate ϕ be computed. The trivial approximation $\phi(p) \equiv 0$ yields the traditional Galerkin approximation; in the context here, it is called the *flat Galerkin method* since Ω is approximated as being "flat." For a *nonlinear Galerkin method* [MT], a nontrivial approximation for ϕ is used, providing a nonlinear manifold instead of a subspace. Approximations to ϕ can be obtained in a variety of ways. Basically, the methods involve either (i) solving a PDE for $\phi(p)$, which is derived from (4a), (4b), or (ii) using (4b) to find an approximate $\phi(p)$, which is then used in (5). See [LS] for a list of references and for a review of different methods, along with some of their theoretical and practical properties.

Methods from class (ii) wherein $q = \phi(p)$ is approximated using (4b) appear to be more straightforward to use in practice than those from class (i), and we restrict our attention to class (ii). In the next few sections, we discuss previously used methods and propose several new variants.

2. Solving the ODE (4b). The most straightforward method of approximating $q = \phi(p)$ is to discretize (4b). Simple low-order methods are normally used. The groundbreaking papers [FJKST] and [FST], which introduce the problem of computing AIMs, use the backward Euler method with one iteration, taking $q = 0$ as the initial approximation. If τ is the stepsize, we simply have

$$q = \hat{\phi}(p) = -\tau(I + \tau AQ)^{-1} QF(p). \tag{7}$$

A nontrivial decision is how to choose n and τ. While a small value of τ gives better accuracy (smaller local truncation error), it only guarantees that the graph of $\hat{\phi}(p)$ is close to Ω if $q = 0$ is a sufficiently accurate initial approximation. If this is not the case, (4a) and (4b) must be integrated in enough steps that (p, q) is sufficiently close to the manifold. Making use of the exponential rate of attraction to Ω and assuming n is chosen such that λ_{n+1} is sufficiently large, the strategy (7) with $\tau \approx 1/\lambda_{n+1}$ can be given some theoretical justification. However, as an illustration of practical difficulties that can arise, we give the following example.

Example 1. Consider the reaction diffusion equation

$$u_t - \nu u_{xx} + u^3 - u = 0, \qquad u(0, t) = u(\pi, t) = 0 \tag{8}$$

with $\nu = (1/2.5)^2 = 0.16$. We have $Au := -\nu u_{xx}$, so if $m = 3$, then $u(x, t) = \sum_{j=1}^{3} \alpha_j(t) \sin(jx)$. In the case $n = 2$, the relationship (7) is found explicitly in [FJKST] as $\alpha_3 = (\alpha_1^3 - 3\alpha_1\alpha_2^2)(\tau/4(9\tau\nu + 1))$. Using the techniques developed in the next section, we can show that a *general* backward Euler inner iteration (to converge from $q = 0$ to the approximation for the first Euler step)

$$q^{(l+1)} = -\tau(I + \tau AQ)^{-1} QF(p + q^{(l)}) \tag{9}$$

is a fixed-point iteration (also called Picard iteration) on

$$\alpha_3 = \frac{-\tau}{4(9\nu\tau + 1)} \{-\alpha_1^3 + 3\alpha_1\alpha_2^2 + [6\alpha_1^2 + 6\alpha_2^2 + 3\alpha_3^2 - 4]\alpha_3\}. \tag{10}$$

The exact Euler-Galerkin solution is the fixed point of (10) (it is easy to see that a fixed point exists for all values of α_1 and α_2); however, ordinary iteration only converges for certain values of α_1 and α_2, depending on ν and τ. This is illustrated in Fig. 1, which shows a cross section of $\alpha_3 = \alpha_3(\alpha_1, \alpha_2)$ for $\alpha_1 = 0.9$. The derivative of the iteration function in (10) at a fixed point α_3 is

$$\frac{-\tau}{4(9\nu\tau + 1)} \{6\alpha_1^2 + 6\alpha_2^2 + 9\alpha_3^2 - 4\} \tag{11}$$

(this derivative must have a modulus less than 1 for convergence), and a simple *necessary* condition for convergence of the Picard iteration is

$$|6\alpha_1^2 + 6\alpha_2^2 - 4| < \frac{4(9\nu\tau + 1)}{\tau}.$$

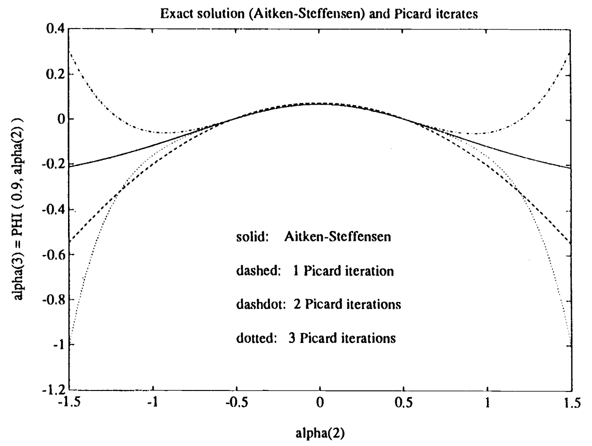


FIG. 1. *Nonconvergence of Picard iteration.*

Figure 2 brings these convergence results under one hat; it shows the region given by the essential convergence condition provided by (11) for $\nu = 0.16$ and $\tau = 1$, and the prediction is confirmed by numerical experiments. This region grows with ν, i.e., added dissipation, but shrinks with increasing τ. Note that in this simple case, the exact solution of (10) can be obtained via an Aitken–Steffensen iteration (see Fig. 1) or by employing a Newton method (see also below). The Euler–Galerkin AIM, as described in [JKT1] and [FST], would actually involve only one iteration of (9) with $m \geqq 6$ when $n = 2$. The lower bound for m is the maximum number of modes which can be stimulated in one iteration of (9) from an initial state $q = 0$. This lower bound depends in an obvious way on the nature of the nonlinearity in $F(u)$. In the case of a polynomial nonlinearity of degree r, for example, the minimum value of m is rn. If k iterations are performed on (9) starting from $q = 0$, the minimum value of m will be $r^k n$. It is shown in [FST] that for n sufficiently large convergence of (9) is guaranteed in an absorbing ball about the solution when there is no truncation of the series (6). On the other hand, our approach throughout is to fix n *and* m a priori and to solve the truncated problem accurately. We shall return to this point later.

Figure 1 in [FJKST] shows a plot of the approximate inertial manifold produced by one Picard iteration of the Euler–Galerkin scheme and the location of five steady-state solutions, with the same parameter values as used here ($\nu = 0.16$, $\tau = 1$). The

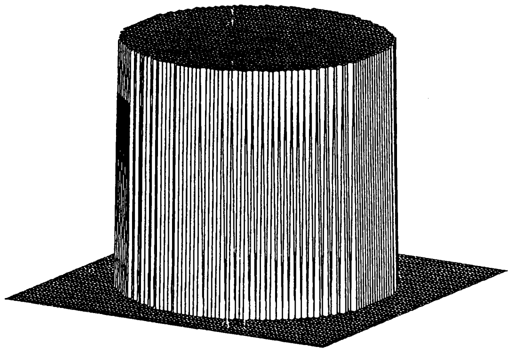


FIG. 2. *Convergence region.*

steady states with $\alpha_1 = 0$ have $\alpha_3 = 0$, and are therefore perfectly reproduced by any iteration scheme starting with initial guess zero. There is a steady state close to the point $\alpha_1 = 1.1143$, $\alpha_2 = 0$, $\alpha_3 = 0.1491$ (which we have computed using a three-mode flat Galerkin approximation). The first Picard iterate for (10) (with $\alpha_1 = 1.1143$, $\alpha_2 = 0$) starting from $\alpha_3 = 0$ produces the value 0.1418, whereas the exact Euler–Galerkin solution is $\alpha_3 = 0.1045$. Thus, while the first iterate overshoots the fairly poor "desired" Euler–Galerkin approximation by around 40 percent, it fortuitously *does* come close to the true steady-state solution. As τ grows without bound, the exact solution of (10) approaches the value $\alpha_3 = 0.1474$, which is indeed close to the desired steady-state value 0.1491, whereas the first Picard iterate overshoots the correct value again, this time giving the poor approximation 0.2402. These results show not only the potential difficulty of choosing a suitable value of τ, but also of using a fixed, small number of Picard iterations.

We shall see that as τ increases, backward Euler and the methods discussed next give virtually identical results, and that for the Kuramoto–Sivashinsky (KS) equation, this in fact happens for most realistic choices of τ.

3. The differential-algebraic equation approach with Picard iteration. An important approach for approximating $\phi(p)$, which is due to Titi [T], involves setting $\dot{q} = 0$ in (4b) and solving

$$G(q) := Aq + QF(p+q) = 0. \tag{12}$$

This defines the so-called *steady approximate inertial manifold* Ω_S with graph $q = \phi_S(p)$. Although not previously interpreted in the literature this way, finding the solution of the (approximate) inertial form (5) with $\phi_S(p)$ is equivalent to solving the semi-explicit differential-algebraic equations (DAEs) (5) and (12). By assumption, the relation $q = \phi_S(p)$ can be determined from (12), so the DAE is of index one and its numerical solution avoids many of the difficulties arising for higher-index problems [AP]. While such an approximation may appear somewhat crude, theoretical arguments can be made for why $\dot{q}$ may be ignored, and the method can give very useful results in practice [JKT1], [JKT2]. One reason for this is because the AIM obtained from solving the algebraic part of this DAE contains all exact steady-state solutions involving only the first m modes (or all those for the flat Galerkin method with m modes), so computations based on the steady AIM give a faithful representation of the steady states for the inertial manifold if one exists for these m modes. In this sense the AIM and the inertial manifold are "threaded together."

The method that has been suggested for solving (12) is the Picard iteration

$$q^{(l+1)} = -A^{-1}QF(p+q^{(l)}) \tag{13}$$

with only one or two iterations and $q^{(0)} = 0$ [JKT1]. Under suitable conditions, justification can be given for this two-iteration case, which gives the so-called *pseudosteady solution* and the corresponding *pseudosteady* AIM $\Omega_{\hat{S}}$, the graph of $q = \phi_{\hat{S}}(p)$. This is done in [JKT1] and [JKT2] for the KS equation. The pseudosteady approach does not involve truncation of the series (6), but since KS involves the quadratic nonlinearity uu_x, truncation of (6) at $m \geqq 4n$ has no effect on the outcome. As with $q^{(0)} = 0$ and k iterations of (13), all components α_j of $q^{(k)}$ (see (6)) with $j > 2^k n$ will be zero. Therefore, (6) may be truncated at $m \geqq 2^k n$. Since k may be large it is convenient to refer to this as the "m unlimited" case.

Recall that our approach is, instead, to fix m and then iterate (13). This approach has several features to recommend it. Since the discrete problem is fixed and independent of the method used to solve it, we need not use Picard iteration and can consider alternatives, such as Newton's method to solve (12). A nontrivial initial approximation for q has no adverse influence; this is particularly beneficial in continuation, where good approximations for q are available. With m fixed, our implementation is not strongly affected by the particular type of nonlinearity. With "m unlimited," on the other hand, m grows rapidly for many types of nonlinearities (e.g., $m=3^k n$ for the Cahn-Hilliard equation, where $F(u)$ involves a u^3 term) and becomes unbounded for others (like $\sin u$). Nevertheless, there is a need to understand the effect of fixing m a priori on the theory of AIMs based upon estimates of the truncation error [JKT1], [T].

Following [JKT1], we take the KS equation in the form

$$u_t + 4u_{xxxx} + \theta[u_{xx} + uu_x] = 0, \qquad 0 \leqq x \leqq 2\pi, \tag{14a}$$

where $\theta \geqq 0$ and with boundary conditions

$$u(x,t) = u(x+2\pi, t), \qquad u(x,t) = -u(2\pi - x, t), \tag{14b}$$

corresponding to the odd case.[2] Note that $A = 4\Delta^2 = 4D^{(4)}$ is the linear dissipation term and the nonlinearity consists of a Burgers nonlinearity uu_x and an antidissipative term Δ. Thus, the eigenvalues and eigenfunctions in (2) are $\lambda_j = 4j^4$ and $w_j = \sin jx$, $j = 1, 2, \ldots$. Note that the linear antidissipative term is not incorporated into the linear operator A.

Introducing the operator notation $B(u,v) = uv_x$, $u, v \in H$, where H is the space of odd 2π-periodic functions in the Sobolev space $H^1(0, 2\pi)$, the Picard iteration (13) takes the form

$$4D^{(4)} \sum_{j=n+1}^{m} \alpha_j^{(l+1)} w_j = -\theta D^{(2)} \sum_{j=n+1}^{m} \alpha_j^{(l)} w_j - \theta Q B(p+q^{(l)}, p+q^{(l)}), \tag{15}$$

where

$$p + q^{(l)} = \sum_{j=1}^{n} \alpha_j w_j + \sum_{j=n+1}^{m} \alpha_j^{(l)}(\alpha_1, \ldots, \alpha_n) w_j.$$

Writing, for convenience, $\alpha_j = \alpha_j^{(l)}$, $j = n+1, \ldots, m$, and taking the projection Q of the convolution sums, it follows that

$$\sum_{j=n+1}^{m} \alpha_j^{(l+1)} w_j = \sum_{j=n+1}^{m} \left\{ \frac{\theta}{4j^2} \alpha_j - \frac{\theta}{8j^4} \beta_j \right\} w_j, \tag{16}$$

where

$$\beta_j = \left\{ \sum_{i=1}^{m-j} (i\alpha_i)\alpha_{i+j} - \sum_{i=j+1}^{m} (i\alpha_i)\alpha_{i-j} + \sum_{i=1}^{j-1} (i\alpha_i)\alpha_{j-i} \right\}$$

are the Fourier coefficients of $2uu_x$. The first two sums combine to give $-j(\sum_{i=1}^{n} \alpha_i \alpha_{j+i} + \sum_{i=n+1}^{m-j} \alpha_i \alpha_{i+j})$. Now we split the third sum into three sums, from 1 to $j-n-1$, from $j-n$ to n, and from $n+1$ to $j-1$ if $j-n-1 \leqq n$ and from 1 to n, from $n+1$ to $j-n-1$, and from $j-n$ to $j-1$ if $j-n-1 > n$, and then combine the first and third of these.

[2] The inertial manifold (IM) has been known to exist for this odd case for some time [FST]; it was recently shown that the KS equation is generally dissipative [I], so existence of the IM follows from the general theory for such systems [FST]. The techniques presented here apply to the more general case, but we restrict attention to the odd case since our purpose is to contrast the numerical results to those obtained previously.

The split-up separates summations into parts that are homogeneous of degrees zero, one, or two in the q-coefficients. Collecting terms, we have

$$\beta_j = -j\left(\sum_{i=1}^{n} \alpha_i\alpha_{i+j} + \sum_{i=n+1}^{m-j} \alpha_i\alpha_{i+j} - \Sigma_3\right) + \Sigma_4, \tag{17}$$

where

$$\Sigma_3 = \sum_{i=1}^{j-n-1} \alpha_i\alpha_{j-i}, \quad \Sigma_4 = \sum_{i=j-n}^{n} i\alpha_i\alpha_{j-i} \quad \text{if } j-n-1 \leqq n$$

and

$$\Sigma_3 = \sum_{i=1}^{n} \alpha_i\alpha_{j-i}, \quad \Sigma_4 = \sum_{i=n+1}^{j-n-1} i\alpha_i\alpha_{j-i} \quad \text{if } j-n-1 > n.$$

The updated iterate values follow directly from (16).

Thus any number of Picard iterations for the KS equation (14) can be computed from (16). The algorithm determines an approximation to $q = \phi_S(p)$, i.e., for fixed n and m, given any value of p we can compute the q components $\alpha_j = \alpha_j(\alpha_1, \alpha_2, \ldots, \alpha_n)$ for $j = n+1, n+2, \ldots, m$, to any accuracy if (16) converges. In § 5 we perform computational experiments to investigate how many Picard iterations can be necessary in practice.

4. The DAE approach with a Newton method. Here we consider Newton and quasi-Newton methods to solve the nonlinear algebraic equations (12). Since

$$\frac{\partial G}{\partial q} = AQ + Q\frac{\partial F}{\partial q}(p+q),$$

Newton iteration gives an update q_{new} to an approximation q_{old} via the relationship

$$[A+J]q_{\text{new}} = -QF(p+q_{\text{old}}) + Jq_{\text{old}}, \tag{18}$$

where $J = Q(\partial F/\partial q)(p+q_{\text{old}})$. The matrix on the left-hand side of (18) is the trailing principal submatrix of dimension $m-n$ for the Jacobian of the full nonlinear system corresponding to (4) with $\dot{p}=0$, $\dot{q}=0$. We denote this full Jacobian by $\bar{A}+(\partial F/\partial\alpha)$ where $\alpha = (\alpha_1, \ldots, \alpha_m)^T$. A calculation along the same lines as used in deriving (17) yields $\bar{A} = 4S^4$, where $S = \operatorname{diag}(1, 2, \ldots, m)$ and

$$\frac{\partial F}{\partial \alpha} = \theta\left(\frac{1}{2}S(T-H) - S^2\right). \tag{19}$$

Here T is the skew-symmetric Toeplitz matrix with first row $(0, -\alpha_1, -\alpha_2, \ldots, -\alpha_{m-1})$, and H is the Hankel matrix with first row $(\alpha_2, \alpha_3, \ldots, \alpha_m, 0)$ and all zeros in the last row.

From (19) we see that the Jacobian has a very simple structure and is easy and cheap to calculate. If $m \leqq 2n+1$, the Jacobian $A+J$ is a *constant* matrix, i.e., (12) is a linear system of equations in q. This particular property of the Jacobian holds whenever the nonlinearity is a product of powers of only two derivatives of u (see [RST], which examines the form of the Jacobian for the spectral method for general nonlinear PDEs).

It is interesting to interpret the Picard iteration (13) in the context of iterative methods for solving systems of linear equations. When J is constant it amounts to a "partial" Jacobi iteration, where (15) is written with the diagonal term $4j^4$ multiplying

$\alpha_j^{(l+1)}$ instead of the full diagonal term $4j^4 - \theta j^2$. A useful study could be to compare this with Jacobi or Gauss-Seidel iteration, although that is not our purpose here.

We end this section with some remarks about how the Picard iteration for (12) compares with the iteration (9) for the backward Euler approximation to (4b) to arrive at an approximation to $q = \phi(p)$. The iteration matrix for (9) is simply

$$\operatorname{diag}\left(\frac{\tau}{4(n+1)^4*\tau+1}, \frac{\tau}{4(n+2)^4*\tau+1}, \dots, \frac{\tau}{4m^4*\tau+1}\right)*J,$$

where $A+J$ is the constant Jacobian in (18). For (13) the iteration matrix is

$$\operatorname{diag}\left(\frac{1}{4(n+1)^4}, \frac{1}{4(n+2)^4}, \dots, \frac{1}{4m^4}\right)*J.$$

It is interesting to note the mathematical similarity between these iterative methods; for the Euler-Galerkin approach the term λ_j in the Picard iteration is replaced by $(1/\tau)+\lambda_j$, which corresponds to a regularization. However, for practical choices of τ—sufficiently large that the theory predicts that we will move close to Ω—$4\tau(n+1)^4$ is generally much larger than 1, and the two iteration processes should produce virtually identical results for the KS equation. This has been our experience in practice, so the Euler-Galerkin results for the KS equation are not given here. Finally, we note that approximating $q = \phi(p)$ by solving the ODE (4b) is related to the basic strategy of applying the Hadamard graph transform [LS], and it is interesting that this approach with an Euler approximation, albeit a simple one, is nearly equivalent computationally to the qualitatively different strategy of solving the DAE for a steady-state approximation.

5. Numerical results. In this section we employ inertial manifold techniques in numerical computations with the KS equation (14). Our interest in the various iterative methods for approximating $q = \phi(p)$ lies more in investigating how well and efficiently they describe the qualitative behaviour of the KS equation as demonstrated in its bifurcation diagram than in presenting detailed computation of the AIMs themselves. Most of our computations use AUTO, a FORTRAN code developed by Doedel [D1] to compute bifurcation diagrams for ODEs (see also [KNS] and [JKT1]). Except where otherwise noted, default parameters are as given in [D2, p. 143]. In particular, solution tolerances are 10^{-4}, although results are frequently verified with smaller error tolerances. All runs are made on a SUN SPARCstation in double precision (roughly 16 decimal digits).

The bifurcation diagrams show plots of the bifurcation parameter θ versus the L^2-norm of the solution. Solid lines correspond to stable steady-state solutions, and broken lines correspond to unstable ones. White squares denote steady-state bifurcation points, and black squares denote Hopf bifurcation points. In the plots for periodic solutions, the time-averaged L^2-norm of the solution is plotted. White circles correspond to unstable periodic solutions, black circles correspond to stable periodic solutions. Period-doubling points are marked with two triangles.

Frequently, the bifurcation diagrams show overlapping branches corresponding to different solutions (with the same L^2-norm). These superimposed branches can have different bifurcation properties. As a result, bifurcation points can be missed if all branches are not searched carefully, and stability assignments for branches can be obscured by their overlap.

To distinguish between the concepts of steady-state solution branches, and "steady" approximate inertial manifolds, we will consistently refer to the former as "steady states," and to the latter as simply "steady." "Pseudosteady" refers to the "steady" AIM approximated by two Picard iterations with "m unlimited."

We will first look at steady-state computations. In interpreting the bifurcation diagrams for the various methods, we must realize that the nonlinear equations arising for the n modes p and $m-n$ modes q nonlinear Galerkin case are *the same* as those for the flat Galerkin method involving m modes, namely

$$APp + PF(p+q) = 0, \qquad AQq + QF(p+q) = 0.$$

In the nonlinear Galerkin method we solve the equations in a way that is rather akin to a nonlinear block Gauss-Seidel method. Starting with an initial guess for p, we determine q by satisfying the second equation and use this value in the first equation to update p.

When we compare the diagrams for an m-mode flat Galerkin method to the ones for an $n+(m-n)$-mode nonlinear Galerkin, then the latter will only show the L^2-norm of the first n Fourier coefficients, whereas the flat Galerkin method will show the L^2-norm of the first m Fourier coefficients. Thus, even if the methods gave identical results, the bifurcation diagrams could differ. But while the bifurcation branches for nonlinear Galerkin tend to lie below those for flat Galerkin, the difference is normally small, as almost all the energy is concentrated in the low modes. Note also that the nonlinear Galerkin method cannot detect bifurcations occurring only in its q-modes.

5.1. Steady-state solutions ($m = 2n$). We consider the problem of computing steady-state solutions for the KS equation. Thus, after approximately solving (12) for $q = \phi(p)$, we compute a bifurcation diagram of steady-state solutions of the ODE (4a) using AUTO. Previously, computations have been done by using either one or two Picard iterations on (12), with the initial guess $q = 0$. We also use the two Picard iteration method (which gave qualitatively the same results as the pseudosteady method in all our examples where those results were accurate) and compare it to approximations to the steady-state solution obtained with Picard iteration (and more than two iterations) or with Newton iteration.

First, we consider the cases $m = 2n = 8$ and 10. For comparison, the "exact" bifurcation diagram is first computed with the flat Galerkin method with 12 modes, and the result is given in Fig. 3. The two Picard iteration results for $m = 8$ and 10 are given in Figs. 4 and 5, respectively. The corresponding steady results, obtained with

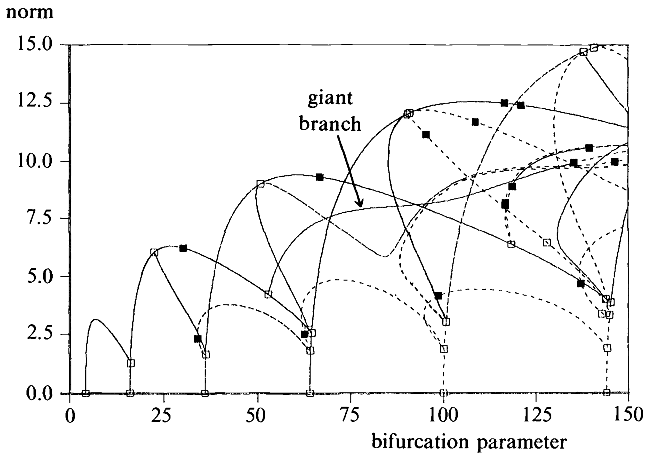


FIG. 3. *Flat Galerkin,* 12 *modes.*

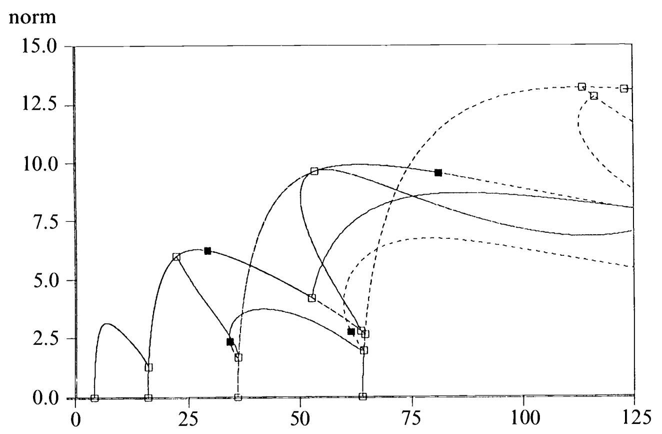


FIG. 4. 4+4 *Nonlinear Galerkin, two Picard iterations.*

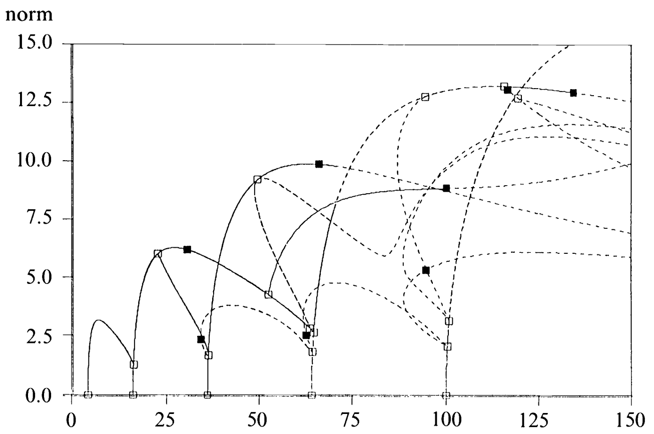


FIG. 5. 5+5 *Nonlinear Galerkin, two Picard iterations.*

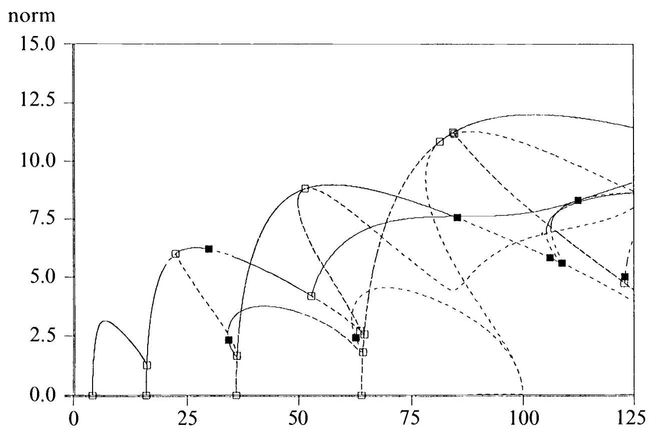


FIG. 6. 4+4 *Nonlinear Galerkin, steady.*

one Newton iteration, are in Figs. 6 and 7.[3] Here and elsewhere, no significant difference in the number of continuation steps used by AUTO was observed for the different flat and nonlinear Galerkin methods. Pseudosteady computations were also done (using truncation at $m=4n$), and the results were not qualitatively different from those in Figs. 4 and 5.

The diagrams in Figs. 3-7 show clearly how the steady solution gives qualitatively better information than the solution computed with only two Picard iterations. It is interesting that useful qualitative information about the steady-state solutions can even be obtained beyond values of θ where an inertial manifold with graph $q=\phi(p)$ exists. The p-variables can no longer be used as coordinates for the manifold, because the assumption that the DAE is index 1 is invalid and hence this relationship does not even hold for all steady-state solutions. Specifically, the term $4(n+1)^4-\theta(n+1)^2$ eventually becomes zero, and for $p=0$ (along the horizontal axis) the diagonal matrix expressing q in terms of p from (12) passes through a singularity (one of the diagonal elements becomes zero). Figures 3, 4, and 6 demonstrate this for $m=8$, where the singularity occurs for $\theta=100$. This raises the point that we may be able to approximate dynamics *locally* with a very low-dimensional manifold—lower than the dimension of any inertial manifold that exists in terms of the basic p-variables—although a safe reliability check of the computations could be difficult to find.

5.2. Steady-state solutions ($m \geqq 2n$). We complete our steady-state computations by considering more numerical results in the case where $m>2n$. We present some results which show that the rate of convergence for Picard iteration can be very slow, in which case the results are bad for large θ. Even for small θ the approximation using two Picard iterations might have only 1 to 2 digits accuracy. It is far from clear when such accuracy is sufficient to correctly capture the qualitative solution information.

Consider data set 1, consisting of $n=4$, $p:=\{4.2182, -3.9175, 0.8192, -0.1037\}$, and $\theta=10$. The solution to (12) is computed for $m=8$, 10, 12, and 16 with Picard and Newton iteration, using $q^{(0)}=0$ and terminating the iteration when

$$\|q^{(i+1)}-q^{(i)}\|_\infty \leqq TOL^* \|q^{(i+1)}\|_\infty, \tag{20}$$

where $\|q\|_\infty$ denotes the Fourier coefficient of q of largest magnitude. For $m=16$, to

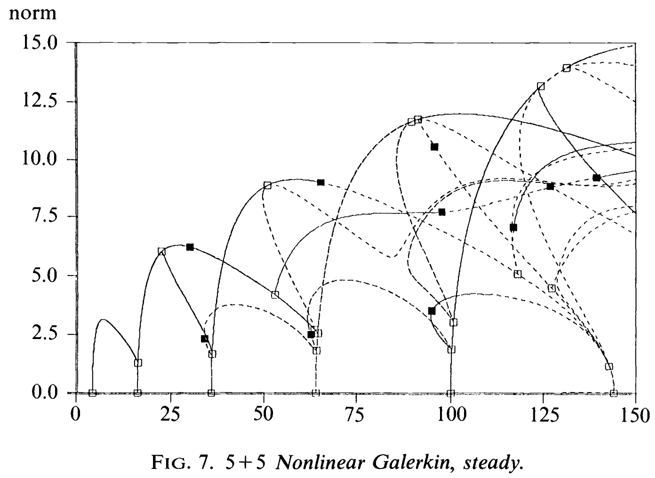


FIG. 7. 5+5 *Nonlinear Galerkin, steady.*

[3] AUTO continuation parameter values are adjusted to DSMIN = .001 and DSMAX = 1 when necessary for obtaining the finer resolution required in the diagrams.

four significant digits, q is {.4021(−1), −.5637(−2), .1019(−2), −.1641(−3), .2381(−4), −.3806(−5), .5521(−6), −.8140(−7), .1180(−7), −.1669(−8), .2365(−9), −.3298(−10)}. Corresponding q-values for the other values of m differ at most in the last digit shown. The relative residual values $\|q^{(i)}-q\|_\infty/\|q\|_\infty$ for the Picard iterates $q^{(i)}$ are .93(−1), .72(−2), .43(−3), .27(−4), .16(−5), .19(−6). In the constant Jacobian case $m=2n$, this compares well with the rate of convergence that analysis would predict, namely, linear convergence with convergence factor $\rho(A^{-1}*J)\approx .084$ ($A^{-1}J$ has eigenvalues $\lambda=.074\pm .04i$, $-.066$, and $-.045$). For the Newton iteration, the relative residual values are .26(−4), .17(−15), ... when $m=16$; for the other values of m, the relative residual is also at roundoff error level after two iterations, with basically the same values. As an estimate of the relative efficiency of Picard versus Newton iteration, we calculate $CPU_P :=$ CPU time for Picard iteration, $CPU_N :=$ CPU time for Newton iteration, and $E=CPU_P/CPU_N$, and values of E are given for various tolerances (TOL) in Fig. 8(a). Note that even though Newton's method converges much faster on an iterate by iterate basis, it is sufficiently expensive for large m that the Picard iteration is more efficient. However, Picard iteration is sufficiently slow that the two-iteration solution has only one digit of accuracy; for example, for $m=8$, $q^{(2)}=${.3992(−1), −.5473(−2), .9118(−3), −.1308(−3)}, and for $m=16$, $q^{(2)}$ has the same first four components and remaining components {.1469(−4), −.1769(−5), .1873(−6), −.1582(−7), .1028(−8), −.4750(−10), .1505(−11), −.2629(−13)}.

Data set 2 consists of $n=8$, $p:=${9.91, 8.87, −7.79, 6.65, 5.53, −4.47, 3.31, 2.29}, and $\theta=10$, and we solve for $m=16$, 18, and 32. The Picard iteration converges fairly rapidly, e.g., for $m=32$ the relative residuals (in the maximum norm) are .25(−1), .62(−3), .19(−4), .96(−6), .25(−7), .92(−9), ... As a result, even though only two Newton iterations are required for $m=18$ and 32 to almost reach roundoff error level, Picard iteration is generally more efficient (cf. Fig. 8(c)). Indeed, the two-Picard iteration solution is quite accurate in this case, e.g., for $m=16$ all except the last coefficient have two to three significant digits.

For large θ, Newton's method becomes more appealing. Taking data set 3 to be the same as data set 1 except that $\theta=100$, the q values decrease at a much slower rate. For $m=16$ they are {.9566, −.4830, .3683, −.1548, .6022(−1), −.2661(−1), .1111(−1), −.5850(−2), .2767(−2), −.1271(−2), .5530(−3), −.2294(−3)}, and q values obtained using lower values of m differ substantially from corresponding components in this vector. For example, with $m=8$, the q components are {.9615, −.4762, .3823, −.1711}. Picard iteration converges much more slowly than before—the eigenvalues of $A^{-1}*J$ are 10 times the values for $\theta=10$—and the two-Picard iteration solution is inaccurate: for $m=16$ it is $q^{(2)}=${.7100, −.1610, .6332(−1), −.1190(−1), .1469(−2), −.9249(−3), .1510(−3), −.1493(−4), .1028(−5), −.4750(−7), .1505(−8), −.2629(−10)}. Still, Newton's method converges to roundoff error levels after four iterations, and its general superiority to Picard iteration in this case, especially for small m, is seen from Fig. 8(b).

While we have run experiments to compare Picard and Newton iteration on a number of data sets, the above are representative of our findings. For $m=2n\leqq 20$, one Newton iteration is cheaper than six Picard iterations. Newton's method has proven very reliable and gives rapid convergence, but it is not always superior to the Picard iteration method. For small θ and large m the rate at which the coefficients of q decay can vary substantially depending upon the value of p, giving fairly slow linear convergence for Picard iteration. However, even then, Picard iteration can be quite competitive with Newton iteration; this is not totally surprising, since the dimensions of the Jacobian matrix (19) become large.

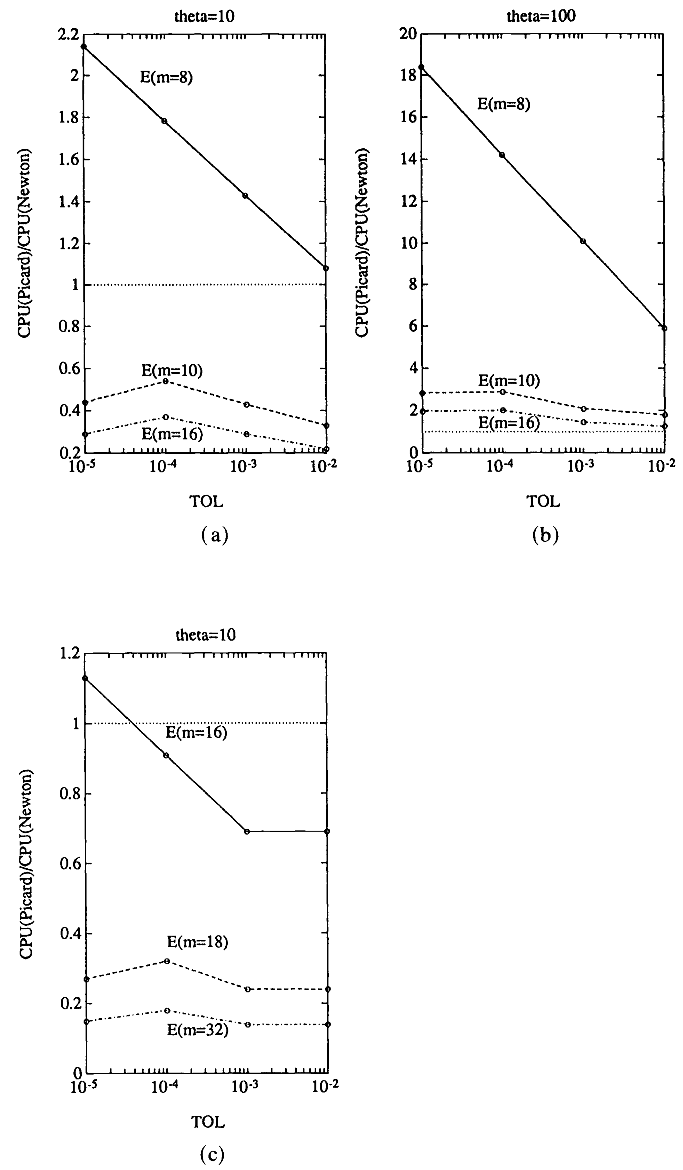


FIG. 8. *Relative efficiency.*

5.3. Steady-state solutions ($m > 2n$), Quasi-Newton methods. The Picard iteration method can be interpreted as a quasi-Newton method obtained by setting $j = 0$ in (18). We have tried other simple quasi-Newton methods when $m > 2n$, which involve choosing only part of J and then performing a frozen Jacobian (fixed-point) iteration. If we use only the components of J which involve p, then the Jacobian approximation consists of the sum of a diagonal and a Toeplitz matrix. This fixed-point iteration turns out to be generally unsatisfactory, since it is not as fast as Picard iteration when the latter works well, and can fail to converge in cases where Picard iteration does not work well. Another strategy is to choose $\hat{m} = 2n + 1$, to solve for the corresponding values $\alpha_{n+1}, \alpha_{n+2}, \ldots, \alpha_{\hat{m}}$, which involves only solving a linear system since the

Jacobian for this problem is constant, and then to approximate the Jacobian (for the $m-n$ coefficients of q) by only using the components of J which involve the first $\hat{m}=2n+1$ coefficients. This method, although again not providing any significant advantage over Picard iteration when the latter converges rapidly, is an efficient fixed-point iteration in other cases. For instance, when $m=16$, $n=4$, and $\theta=10$, the first iterate for q is .4021(−1), −.5637(−2), .1019(−2), . . . , and the relative residual reaches roundoff error level after one iteration for the complete q. For $\theta=100$, six iterations are needed.

5.4. Giant branch computations. We now compare some of the iterative methods for computing steady-state solutions along the so-called *giant branch* or *coherent structure branch*, which is labeled in Fig. 3. For discussion of the importance of the giant branch solutions, we refer to [JKT2]. What is of interest here is to approximate this giant branch with moderately small values of n and to have the computed solution preserve the dissipation in the PDE. In the bifurcation diagram, preservation of dissipation is related to the computed giant branch solution growing in magnitude but not switching back toward the vertical axis. For $\theta \leqq \theta_0(n)$ the flat Galerkin solutions preserve dissipation, as do the steady solutions for the nonlinear Galerkin method, but the pseudosteady solutions do not have this property [JKT2].

In order to compute nonlinear Galerkin solutions that do preserve dissipation, Jolly, Kevrekidis, and Titi [JKT2] use several approaches. One involves a "pseudo-Euler–Galerkin" approximation formed by dropping one term from the discretized equations. Another involves attempting to "prepare" the PDE by dropping terms. The latter proves less than ideal because the qualitative properties of the prepared PDE can differ significantly from those of the original PDE, but this seems difficult to predict from the analysis. In general, it appears difficult to analytically predict when and/or if the computed giant branch solutions accurately describe the actual ones for these modified methods.

Our approach here is to verify that the steady solution (with m fixed) preserves dissipation, and thereby to emphasize the importance of solving the discretized equation for this solution *accurately*. (This solution is equivalent to the m-mode flat Galerkin solution, which is known to preserve dissipation [JKT2].) Solving (12) exactly is easy since the Jacobian is constant when $m=2n$, and only one Newton iteration is required. The results are compared with those obtained with flat Galerkin directly.

The flat Galerkin solutions for $m=6$, 12, and 24 are shown in Figs. 9(a), 9(b), and 9(c), respectively. The flat Galerkin results for $m=40$ are virtually identical to those for $m=24$ except for a slight shift in the location of the second Hopf bifurcation point. The nonlinear Galerkin solutions for $m=2n=6$ are depicted in Figs. 10(a)–10(c) for the cases of 2 and 25 Picard iterations and for Newton's method.[4] These solutions show what typically happens for a fixed number of Picard iterations. The solutions start rising rapidly at some point, and there are spurious Hopf bifurcations near the region of breakdown. These indicate the appearance of complex eigenvalues, which probably arise from onset of ill-conditioning of the nonlinear systems (however, this point deserves further study). Working with a fixed number (even 25) of Picard iterations eventually becomes unreliable as θ increases, although in this particlular case the giant

[4] The results from two Picard iterations in Fig. 10(a) are unchanged when solution tolerances are reduced to 10^{-7}. Interestingly, the corresponding pseudosteady solution branch originates from the origin in the same way but then does *not* make the turn back to the right, continuing instead to grow without bound as it approaches the vertical axis [JKT2]. Thus, two iterations seem to be insufficient regardless of m.

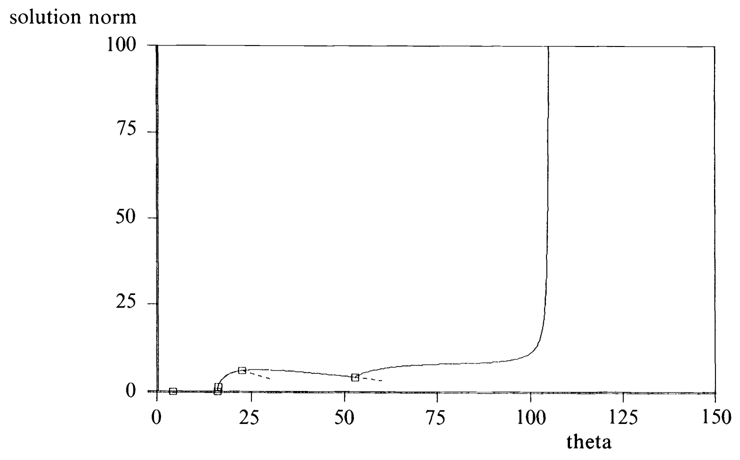


FIG. 9(a). *Flat Galerkin, six modes (giant branch).*

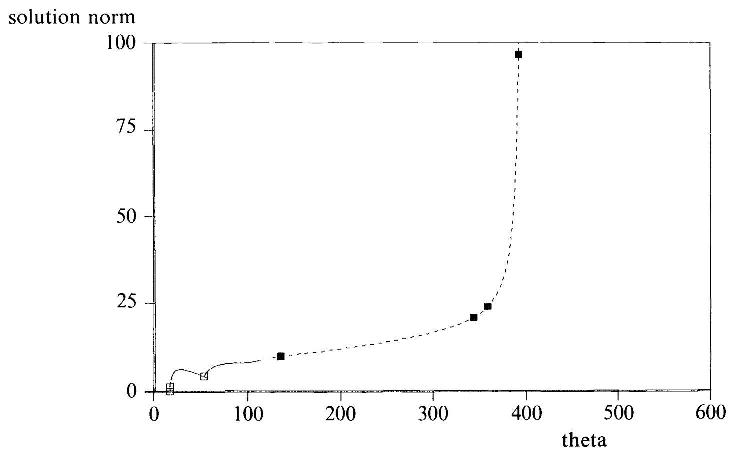


FIG. 9(b). *Flat Galerkin,* 12 *modes (giant branch).*

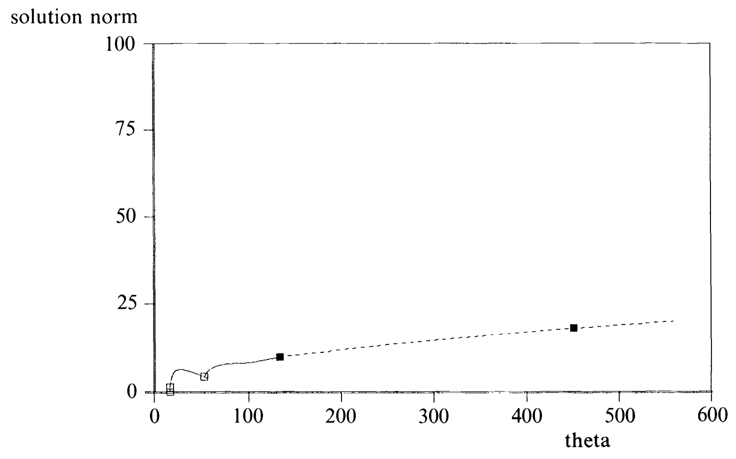


FIG. 9(c). *Flat Galerkin,* 24 *modes (giant branch).*

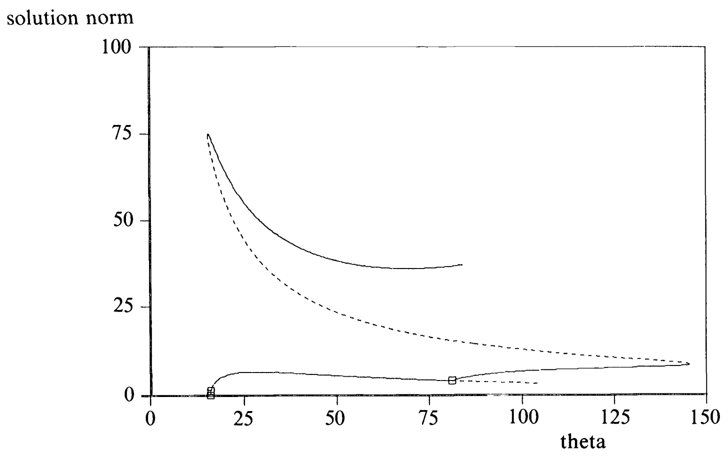


FIG. 10(a). 3+3 *Nonlinear Galerkin, two Picard iterations.*

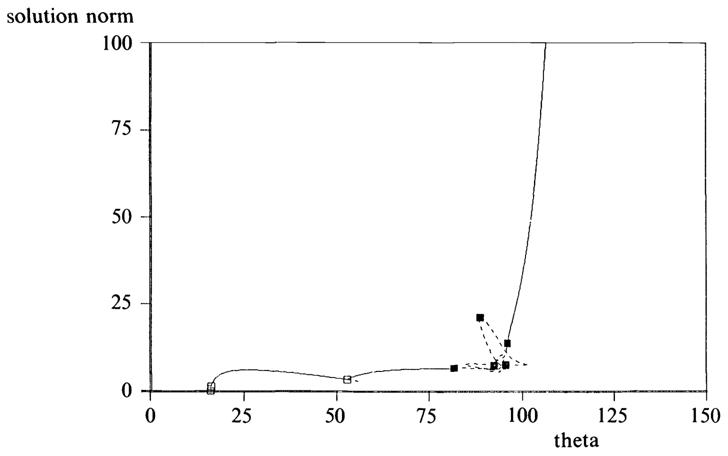


FIG. 10(b). 3+3 *Nonlinear Galerkin,* 25 *Picard iterations.*

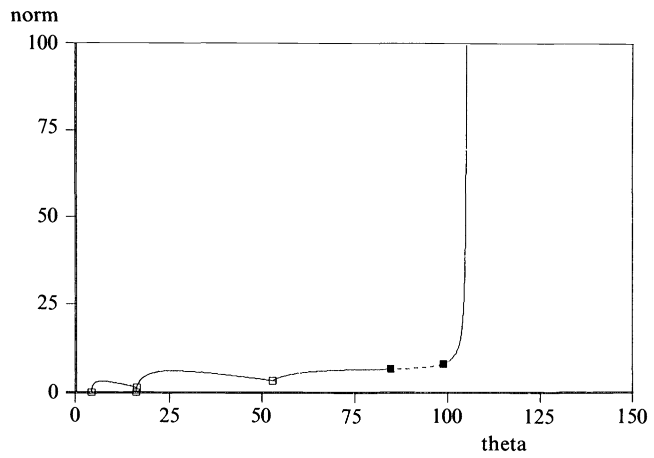


FIG. 10(c). 3+3 *Nonlinear Galerkin, steady* (*Newton*).

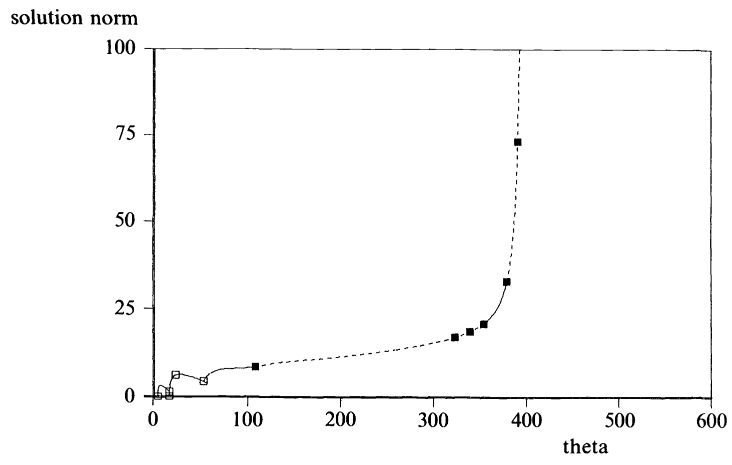


FIG. 11(a). 6+6 *Nonlinear Galerkin, steady (Newton).*

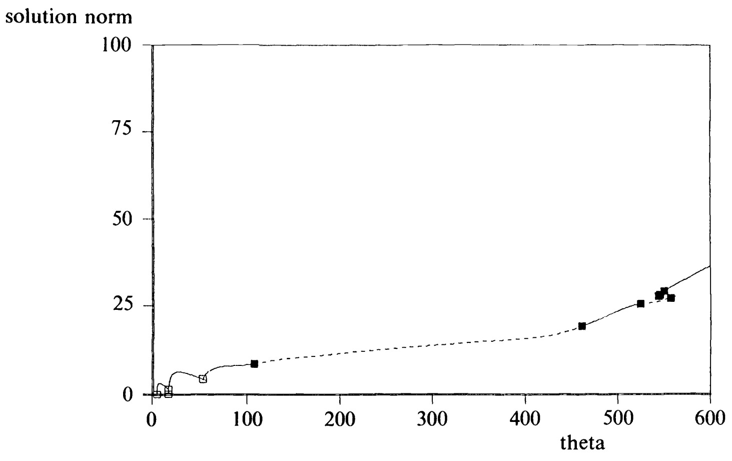


FIG. 11(b). 6+18 *Nonlinear Galerkin, steady (Newton).*

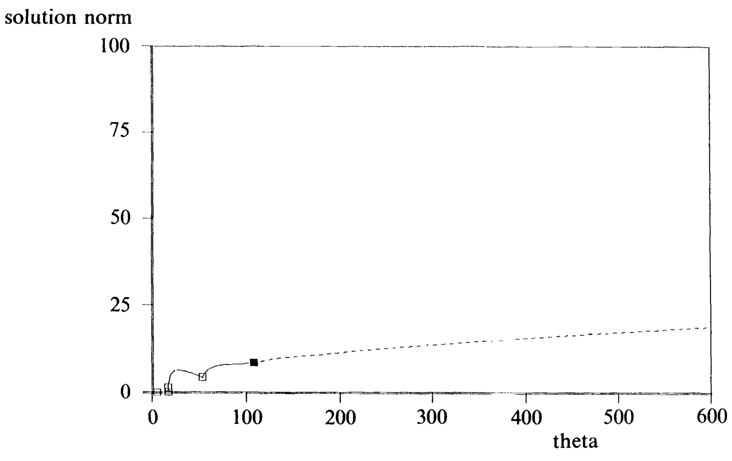


FIG. 11(c). 6+18 *Nonlinear Galerkin, steady (Newton),* TOL = 1E − 6.

branch computations do eventually diverge like the ones for the steady solution and the flat Galerkin solution with $m=6$.

The steady solutions for $m=2n=12$ are shown in Fig. 11(a). Again, the results demonstrate clearly that the steady solutions have the nice dissipation properties of the flat Galerkin solution and that the solution breakdown is characterized by the appearance of a large number of spurious Hopf bifurcations.

Finally, we choose n small and investigate how much qualitative information can be extracted from the steady solution as compared to the corresponding flat Galerkin solution with $2n$ modes. Figure 11(c) shows the results from computing the steady solution with Newton's method for $m=3n=18$. While the first Hopf bifurcation point is shifted somewhat and in the absence of higher modes the second one is not detected, the qualitative information about the giant branch is otherwise accurate. Even though an inertial manifold cannot exist globally past $4(n+1)^2=196$, the local qualitative information around the giant branch is accurate over the parameter range [0,600]. This is to be contrasted with previous cases when $n=6$.

If the initial approximation $q=0$ is used, then Newton's method does not converge for values around $\theta=450$. To obtain the results in Fig. 11(b) we use the solution q from the previous parameter value θ as an initial approximation q for a new value of θ. This is a natural choice for the continuation process, and allows for a speedier computation. Even if this is done and three Newton iterations are used in each continuation step, then trouble arises around $\theta=450$, where spurious Hopf bifurcations enter the picture.

5.5. Periodic solutions. We finish this section with computations of *nonsteady-state* solutions, viz., periodic solutions branching from Hopf bifurcation points. We limit consideration to two Hopf bifurcation points, located near (34.25, 2.3) and (30.3, 6.2) on the $(\theta, \|u\|)$ plane. Results are limited to a rough computational comparison of flat Galerkin with $m=10$ and nonlinear Galerkin with $m=2n=10$. Default parameter values are again used in AUTO unless otherwise specified. The nonlinear Galerkin solution branches emanating from the two Hopf bifurcation points are shown in Fig. 12.

Finer detail of the periodic solutions generated from the lower Hopf bifurcation point with the nonlinear Galerkin method is seen in Fig. 13. One period doubling is

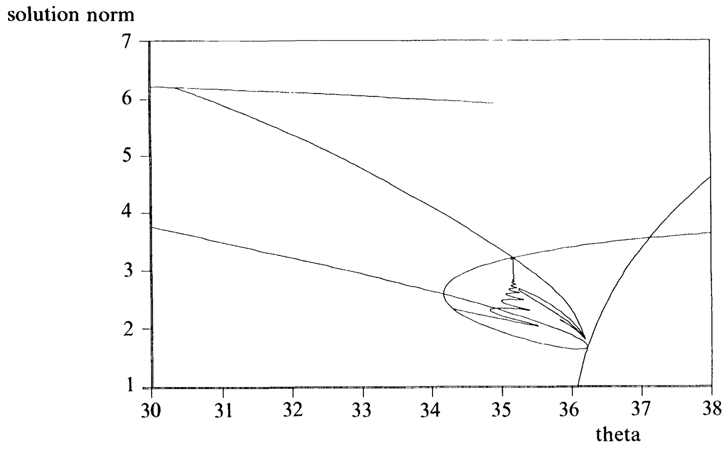


FIG. 12. 5+5 *Nonlinear Galerkin, six Picard iterations.*

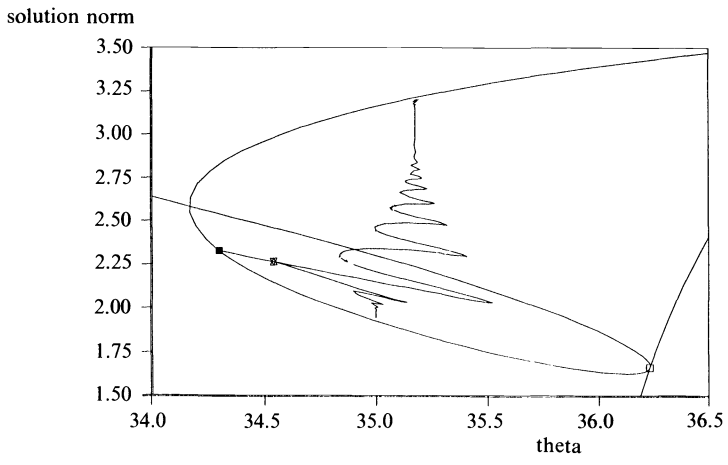


FIG. 13. 5+5 *Nonlinear Galerkin, six Picard iterations.*

found, and the corresponding periodic solution branch also computed.[5] Both branches consist entirely of unstable periodic solutions. These two periodic branches indicate Silnikov-type homoclinic trajectories approaching steady-state solution branches. In general, we have observed that the flat Galerkin computations require more computer time than those for nonlinear Galerkin (with the same m and $m=2n$), and this difference is particularly significant for the periodic solutions. We have also computed pseudosteady solutions with $m=4n=20$ in order to compare with the other nonlinear Galerkin results. As in the steady-state solution case, no improvement has been observed over the corresponding method with m fixed ($m=2n=10$, and two Picard iterations).

A blowup of part of the periodic solution branch generated from the top Hopf bifurcation point with the nonlinear Galerkin method is shown in Fig. 14. The flat Galerkin method runs into considerably more difficulty following this branch, although further study would be required before we could predict whether or not this is to be

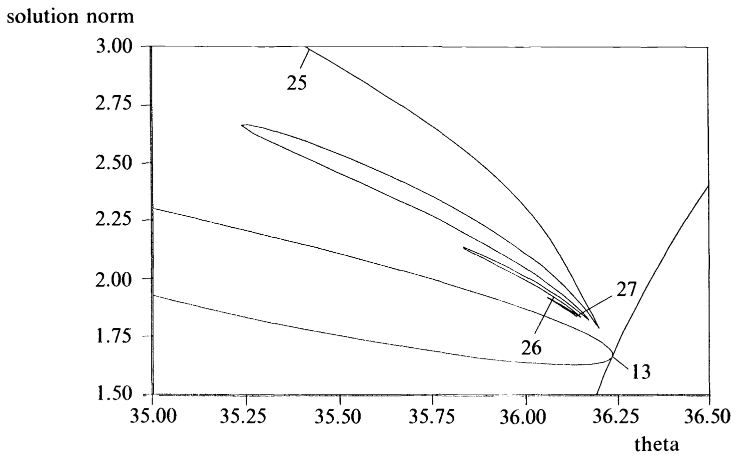


FIG. 14. 5+5 *Nonlinear Galerkin, Newton.*

[5] See [GS] for a description of a richer underlying structure of periodic solutions, much of which can be recovered using more careful continuation with AUTO.

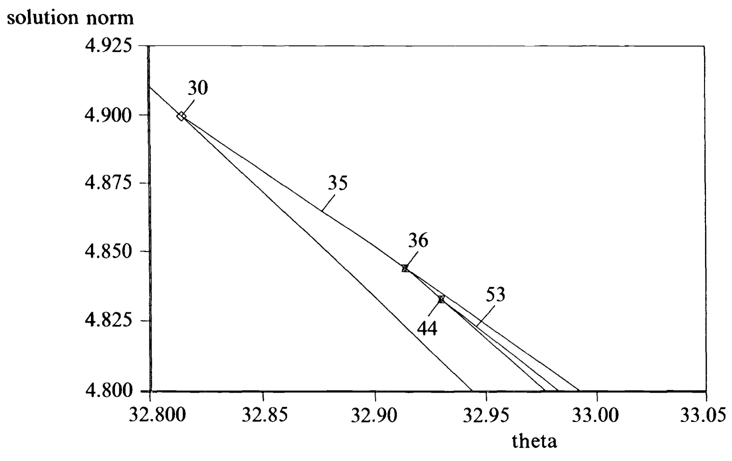


FIG. 15(a). 5+5 *Nonlinear Galerkin, Newton.*

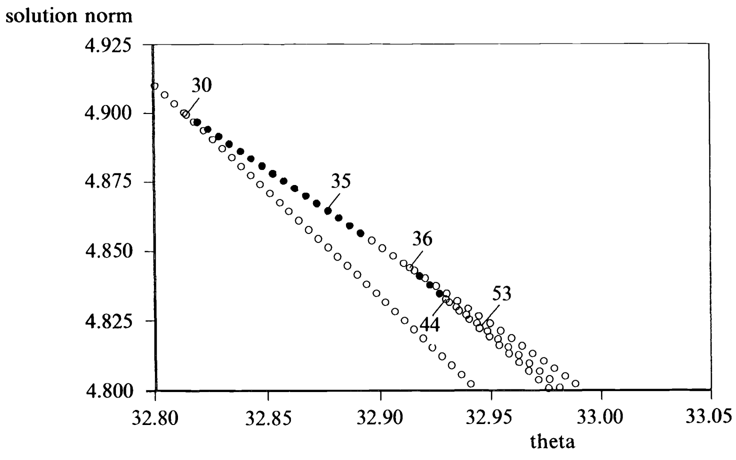


FIG. 15(b). 5+5 *Nonlinear Galerkin, Newton.*

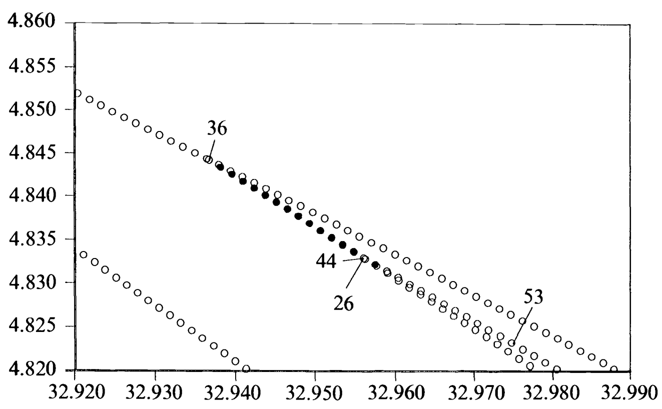


FIG. 15(c). 5+5 *Nonlinear Galerkin, Newton.*

expected. Finer detail of the nonlinear Galerkin results are shown in Figs. 15(a)–15(c). The periodic solutions emerging from the top Hopf bifurcation point as θ increases begin stable, but lose stability at a bifurcation point around (32.82, 4.9). The results displayed in Fig. 15 suggest that at this point, stable and unstable periodic solutions are generated. The stable periodic solution loses stability at a period-doubling point around (32.92, 4.85). The stable period-doubled branch becomes unstable at a second period-doubling point around (32.93, 4.84). At this latter point both emerging periodic branches appear to be unstable, and no further close-by period doublings are observed (with the default parameter values for AUTO). The corresponding flat Galerkin method finds another bifurcation point on the periodic solution branch nearby the top Hopf bifurcation point but misses the second period-doubling point. The location of this second period-doubling point was verified with a nonlinear Galerkin method using $p = q = 7$. It is only one of a wealth of period-doubling and bifurcation points in this region [SW]. It is of interest to note that bifurcations from the top Hopf bifurcation point have been calculated and discussed in [JKT1]. Their calculations reveal, inter alia, a sequence of period doublings, with stability inherited by the period-doubled branch at each bifurcation. They also make the point that details of the periodic solutions may be captured using a variety of three-mode AIMs.

Indeed, it is important to emphasize that results can quickly become sensitive and bifurcation points may be missed (see Doedel's ample description in [D1]), and more careful study requires changing parameter values in AUTO (e.g., setting smaller tolerances or increasing the number of subintervals for spline collocation). This in turn can greatly increase computation time, particularly for periodic solution branches. Nevertheless, the fine detail given by the nonlinear Galerkin approximation is impressive.

As an independent test of the reliability of these results, we have taken output from AUTO as initial data for a numerical integration of (14) with respect to time. The Fourier coefficients corresponding to particular points shown in the bifurcation diagram in Fig. 15 are obtained using a nonlinear Galerkin method based on a Newton iteration. These coefficients are used as initial conditions for the nonlinear dynamical system arising from the spectral discretization of (14), and this system is then integrated in time using a standard routine (ode23 of MATLAB). In particular, we investigate the time evolution of the periodic solutions corresponding to labels 36, 44, and 53 in Fig. 15. The corresponding θ values are 32.93678, 32.95609, and 32.97492. The stability of the solution corresponding to label 36 is evident from Fig. 16(a), which displays the results of integrating over six periods (results over 60 integration periods are not qualitatively different). Nevertheless, the solution's stability region is small, as perturbation of the initial data by 1 percent causes the integration to converge to a stable steady-state solution on the branch above this one (in the $(\theta, \|u\|)$-plane); see Figs. 16(a), 7, and 12 for the location of the branches in the bifucation diagram. The initial data for the solution corresponding to label 44 appears to be virtually identical to that for label 36, varying in only the fourth decimal place (and thus of the same order as the 10^{-4} tolerances used in AUTO to compute these solutions). Nevertheless, the period-doubled solution in t is evident in Fig. 16(b) (the label 36 and label 44 solutions are superimposed). AUTO estimates only one Floquet multiplier greater than unity in magnitude, and it only barely so. This is consistent with the numerical integration which shows that the solution is moderately unstable at worst. The solution corresponding to label 53, with a Floquet multiplier approximately 12 in magnitude, indeed displays instability in numerical integration (results not shown). Fig. 16(c) shows the initial solution profile for label 36 (the graph for the label 44 solution is indistinguishable

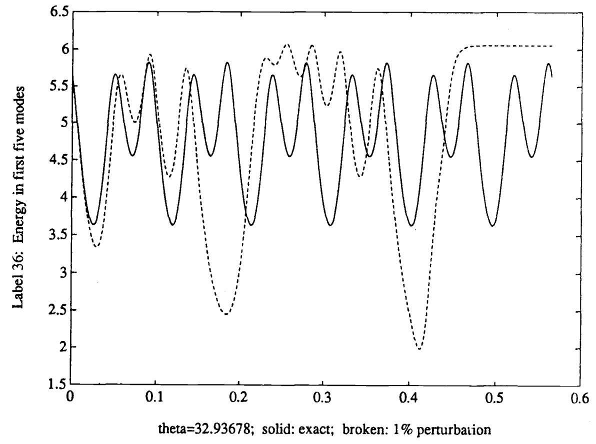


FIG. 16(a). *Time integration over six periods* (5+5 *Newton-Galerkin*).

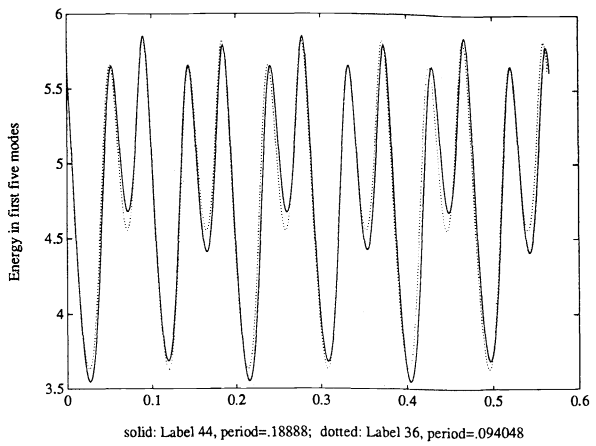


FIG. 16(b). *Period doubling* (5+5 *Newton-Galerkin*).

from the one for label 36), and Fig. 16(d) gives the time evolution of $u(x, t)$ over three periods.

6. Conclusions. During the last few years the study of inertial manifolds for dissipative PDEs has been a very active area of research. One of the principal approximations of these manifolds is the steady AIM. To date, numerical techniques for their computation have been limited to the use of at most two cycles of a Picard iteration. Here we have shown that solving the governing nonlinear equations by this rather limited technique could lead to erroneous results. Numerical experiments using AUTO [D1] have shown, inter alia, that steady-state bifurcation diagrams are computed more accurately if a general Picard iteration (more than two iterations) is used.

We have also demonstrated that Newton's method is a viable and valuable tool for the computation of a steady AIM. For the KS equation, for example, where the nonlinearity is quadratic, the Jacobian has been shown to be constant, with a particularly simple structure, if the number of high modes does not exceed the number of low modes by more than one. A two-level quasi-Newton method has also been introduced, and evidence indicates that this approach is of value in certain circumstances. This

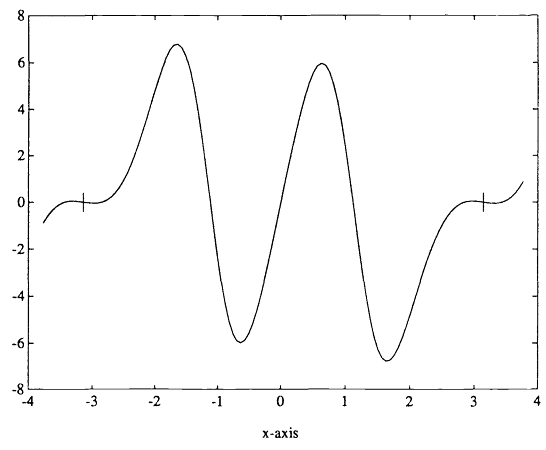


FIG. 16(c). *Snapshot* $u(x, 0)$ *for periodic solution at* $\theta = 32.95609$.

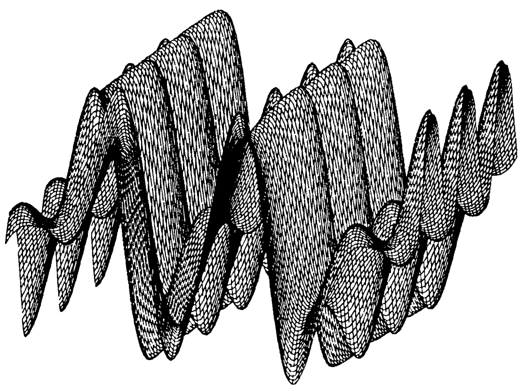


FIG. 16(d). *Time evolution* $u(x, t)$ *over three periods.*

two-level method is readily extended to give a multilevel scheme: the use of such a scheme will be examined in situations where the dissipative effects are less pronounced than they are in the KS equation.

The usefulness of the AIM for the KS equation has been established. When this AIM is computed accurately with a Newton or Picard iteration, the dissipative properties of low-dimensional AIMs are preserved without recourse to techniques such as modifying the PDE itself. Good qualitative approximations to the giant branch are found at relatively little cost. Furthermore, fine-scale qualitative approximations to nonsteady-state solutions are also accurately computed using a low number of modes.

An important departure from previous approaches for computing AIMs has been the replacement of the projection Q of infinite rank by one of finite rank. Fixing the number of q-modes permits the use of arbitrary initial approximations, an often essential and in any event efficient process when doing continuation to construct bifurcation diagrams. Also, the implementation is then not as strongly affected by the type of nonlinearity appearing in the PDE.

This approach of fixing the discretization parameters n and m has additional advantages. It allows us to check the accuracy of the two-Picard iteration method, which as the theory nicely predicts, is satisfactory for moderate values of θ but is insufficient as θ grows. Yet even though there is no AIM (global manifold capturing the dynamics of the inertial form) of dimension n for a range of values of θ, we have seen that we can still compute reasonable local approximations (or local manifolds). We need to develop a theory to better understand the nonlinear Galerkin approach with m fixed in this way and to guide the selection of both n and m.

Investigations of inertial manifolds have thus far dealt with cases where the exact eigenfunctions of the linear dissipative operator A of the PDE are known. In situations involving more than one space dimension and irregular domains, this is unlikely to be the case, and we will have to resort to space discretization. We have performed numerical experiments in one space dimension with finite differences and the pseudo-spectral method. The results are accurate to the "expected" order of such approximations, but predictably, this approach is computationally less efficient than the use of exact projection techniques. An area open for investigation is the study of the viability of inertial manifolds for problems in dissipative dynamics for which exact eigenfunctions are not available. One of the ultimate goals is the long-time solution of the Navier–Stokes equations in such a situation using a finite number (hopefully small) of nonlinear ODEs. Preliminary steps have been made by Jauberteau, Rosier, and Temam [JRT] in the search for this elixir.

Acknowledgements. We would like to thank Edriss Titi for sharing his enthusiasm for inertial manifolds with us, and for keeping us abreast of theoretical developments in the field. We are also indebted to Lixin Liu, who was a valuable source of AUTO expertise. Finally, we are grateful to Mike Jolly and Yannis Kevrekidis, whose helpful comments have led to improvements in the original version of this work.

REFERENCES

[AP] U. Ascher and L. R. Petzold, *Projected implicit Runge–Kutta methods for differential-algebraic equations*, SIAM J. Numer. Anal., 28 (1991), pp. 1097–1120.

[CFNT] P. Constantin, C. Foias, B. Nicolaenko, and R. Temam, *Spectral barriers and inertial manifolds for dissipative partial differential equations*, J. Dynamics Differential Equations, 1 (1989), pp. 45–73.

[D1] E. J. Doedel, AUTO: *A program for the automatic bifurcation analysis of autonomous systems*, Cong. Num., 30 (1981), pp. 265–284. Also in Proc. 10th Manitoba Conf. on Numer. Math. and Comp., Univ. of Manitoba, Winnipeg, Canada, 1980.

[D2] ———, AUTO: *Software for continuation and bifurcation problems in ordinary differential equations*, California Institute of Technology Applied Mathematics Rep., Pasadena, CA, 1986.

[FJKST] C. Foias, M. S. Jolly, I. G. Kevrekidis, G. R. Sell, and E. S. Titi, *On the computation of inertial manifolds*, Phys. Lett. A, 131 (1988), pp. 433–436.

[FST] C. Foias, G. R. Sell, and E. S. Titi, *Exponential tracking and approximation of inertial manifolds for dissipative equations*, J. Dynamics Differential Equations, 1 (1989), pp. 199–224.

[I] J. Il'yashenko, *Global analysis of the phase portrait for the Kuramoto–Sivashinsky equation*, IMA preprint #665, Inst. for Mathematics and its Applications, Univ. of Minnesota, Minneapolis, Minnesota.

[JRT] F. Jauberteau, C. Rosier, and R. Temam, *A nonlinear Galerkin method for the Navier–Stokes equations*, in Proc. Conf. Spectral and High Order Methods for PDEs, ICOSAHOM '89, Como, Italy, 1989.

[JKT1] M. S. Jolly, I. G. Kevrekidis, and E. S. Titi, *Approximate inertial manifolds for the Kuramoto–Sivashinsky equation: Analysis and computations*, Phys. D, 44D (1990), pp. 38–60.

[JKT2] ———, *Preserving dissipation in approximate inertial forms*, J. Dynamics Differential Equations, 3 (1991), pp. 179–197.

[KNS] I. G. Kevrekidis, B. Nicolaenko, and C. Scovel, *Back in the saddle again: A computer assisted study of the Kuramoto–Sivashinsky equation*, SIAM J. Appl. Math., 50 (1990), pp. 760–790.

[LS] M. Luskin and G. R. Sell, *Approximation theories for inertial manifolds*, Math. Model. Numer. Anal., 23 (1989), pp. 445-461.

[MT] M. Marion and R. Temam, *Nonlinear Galerkin methods*, SIAM J. Numer. Anal., 26 (1989), pp. 1139-1157.

[RST] R. D. Russell, D. M. Sloan, and M. R. Trummer, *On the structure of Jacobians for spectral methods for nonlinear* PDEs, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 541-549.

[SW] J. W. Swift and K. Wiesenfeld, *Suppression of period doubling in symmetric systems*, Phys. Rev. Lett., 32 (1984), pp. 705-708.

[T] E. S. Titi, *On approximate inertial manifolds to the Navier-Stokes equations*, J. Math. Anal. Appl., 149 (1990), pp. 540-557.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 44-48, January 1993

FAST SOLUTION OF NONLINEAR POISSON-TYPE EQUATIONS*

BRETT M. AVERICK† AND JAMES M. ORTEGA‡

Abstract. For certain nonlinear Poisson-type equations, it is possible to make a change of variable so that the solution is obtained by solving a Poisson equation followed by solving one-dimensional nonlinear equations. For those problems for which a Fast Poisson Solver may be used, this method is considerably faster than methods that solve a discretization of the original equation by Newton-type methods. Moreover, there is excellent parallelism in the solution of the one-dimensional equations. Numerical results are given for some model problems on a CRAY-2 and comparisons are made with other methods.

Key words. Poisson equation, nonlinear equations, parallel

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

1. Introduction.

In [3], we considered methods for the numerical solution of nonlinear Poisson-type equations of the form

$$\nabla \cdot [K(u)\nabla u] = f, \tag{1.1}$$

where K is a positive differentiable function. If (1.1) is discretized by finite difference methods, we obtain a discrete system of the form

$$F(\mathbf{u}) = A(\mathbf{u})\mathbf{u} - \mathbf{b}(\mathbf{u}), \tag{1.2}$$

where the matrix $A(\mathbf{u})$ is symmetric positive definite for all $\mathbf{u}$. Newton's method applied to (1.2) is

$$F'(\mathbf{u}^k)\delta^{k+1} = -F(\mathbf{u}^i), \qquad k = 0, 1, \dots, \tag{1.3}$$

where the Jacobian matrix is of the form

$$F'(\mathbf{u}) = A(\mathbf{u}) + B(\mathbf{u}). \tag{1.4}$$

The Jacobian matrix has the usual sparsity of Poisson problems, and we would like to solve the Newton systems (1.3) by a conjugate gradient method. The matrix $B(\mathbf{u})$ in (1.4) is not symmetric, however, which would require using methods such as GMRES [8] for nonsymmetric systems.

An alternative approach considered in [3] is based on the fact that $\|B(\mathbf{u})\|$ is small compared with $\|A(\mathbf{u})\|$ in many cases. This motivated the use of the approximate Newton method

$$A(\mathbf{u}^k)\delta^{k+1} = -F(\mathbf{u}^k), \qquad k = 0, 1, \dots, \tag{1.5}$$

in which the systems of (1.5) are solved by the Incomplete Cholesky Conjugate Gradient (ICCG) method.

In the present paper, we consider an entirely different approach based on the formulation [4] of (1.1) as

$$\nabla^2\phi(u) = f. \tag{1.6}$$

* Received by the editors September 9, 1991; accepted for publication (in revised form) March 20, 1992.

† Army High Performance Computing Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, Minnesota 55415. The research of this author was supported in part by Army Research Office grant DAALO3-89-C-0038.

‡ Department of Computer Science, University of Virginia, Charlottesville, Virginia 22903. The research of this author was supported in part by National Aeronautics and Space Administration grant NAG-1-1112-FDP.

If ϕ is a function such that

$$\phi'(u) = K(u), \tag{1.7}$$

then

$$\nabla^2\phi(u) = \nabla \cdot (\phi'(u)\nabla u) = \nabla \cdot (K(u)\nabla u), \tag{1.8}$$

and (1.6) is equivalent to (1.1). Thus, we can obtain the solution of (1.1), in principle, by a two-stage process: I. Solve the Poisson equation

$$\nabla^2 w = f. \tag{1.9}$$

II. Solve the one-dimensional nonlinear equations

$$\phi(u_P) = w_P, \tag{1.10}$$

where w_P denotes the solution of (1.9) at a point P in the domain.

In the actual algorithm we consider, the domain is first discretized so that (1.9) becomes a discrete Poisson equation, and w_P is the solution of this discrete problem at grid point P. Under the assumption that $K(u)$ is positive, solutions of (1.10) will be unique. If we assume further that $K(u)$ is bounded away from zero, then a solution of (1.10) will exist for any w_P.

Even on conventional machines this approach may have considerable merit since it allows the use of fast Poisson solvers whenever the domain is suitable. Results given in the next section show that for a small two-dimensional problem run on a Sun 3/60, this new method is considerably faster than the best method considered in [3]. On parallel and vector computers, the advantage is even greater since the solution of the nonlinear equations (1.10) has excellent parallelism (or vectorization) across the grid points. In the next section, we will give numerical results for a fairly large (250,000+ unknowns) three-dimensional problem considered in [3]. Comparison to the results in [3] on a CRAY 2 shows considerable superiority of the new method. The method does have limitations, however. These are given in § 3, along with further discussion of parallelism and other properties.

2. Numerical results. We first consider results on a serial computer (a Sun 3/60) for a two-dimensional problem of the form (1.1) on the unit square and with $K(u) = .33 + .91u$; this problem was considered in [3]. We use the Dirichlet boundary conditions $u(x, y) = x^2 + y^2$ and choose the forcing function f of (1.1) so that $x^2 + y^2$ is the exact solution of (1.1). The boundary conditions for the Poisson equation (1.9) are given by

$$w_\Gamma = \phi(u_\Gamma), \tag{2.1}$$

where w_Γ and u_Γ are values of w and u on the boundary Γ, respectively.

We discretize the Poisson equation (1.9) by the usual five-point finite differences on an $N \times N$ mesh of interior grid points. The discrete Poisson problem is then solved by a fast Poisson solver (FPS). Since K is linear, ϕ is quadratic and the solutions of (1.10) are easily evaluated. However, for consistency with more general equations, we used a Newton iteration for these one-dimensional equations. Timing results using double-precision arithmetic are given in Table 1 for $N = 31$ and $N = 63$. The second

TABLE 1
Times (seconds) for two-dimensional problem on a Sun 3/60.

N	FPS	NL	Total	TANICCG	Speedup
31	4.2	1.7	5.9	79.8	13.6
63	23.5	7.2	30.7	669.3	21.8

column gives times for the FPS. NL is the time for solving the nonlinear equations (1.10). For comparison, Table 1 also lists times for the same problem using the TANICCG method from [3]. This method solves the approximate Newton systems (1.5) by ICCG(0), using truncation in the sense of [5] to determine the number of inner iterations; further details may be found in [3]. The last column of Table 1 gives the ratios of the TANICCG times to those of the new method.

We next consider a three-dimensional problem treated in [3] in which

$$K(u)=\frac{(100+27u)^{\frac{3}{2}}}{300+27u}, \tag{2.2}$$

so that

$$\phi(u)=209.6\tan^{-1}(.135u+.5)^{\frac{1}{2}}+(3u+11.1)^{\frac{1}{2}}(2u-37). \tag{2.3}$$

The domain is the unit cube, and the forcing function f of (1.1) was chosen so that $x^2+y^2+z^2$ is the exact solution of the differential equation. Thus, the boundary values are also given by $x^2+y^2+z^2$, and then the boundary values for the Poisson equation (1.9) are obtained from (2.1).

Table 2 gives timings for single-precision arithmetic on a single processor of a CRAY 2 for $N=31$ (29,791 equations) and $N=63$ (250,047 equations). As in Table 1, the second column is the time for the FPS, and NL is the time for solving the nonlinear equations (1.10). These one-dimensional nonlinear equations were solved by Newton's method, vectorized across all the grid points.

In [3], we used for (1.5) the initial approximation

$$u^0_{i,j,k}=x_i+y_j^2+z_k^2, \tag{2.4}$$

which is a linear interpolation of the boundary values in the x-direction. For the Newton iterations for (1.10), we used (2.4) and we also chose $u^0_{i,j,k}$ as the solution of the discrete Poisson problem at the i, j, k grid point. Of these two ways to choose $u^0_{i,j,k}$, (2.4) was the best and only these times are reported in Table 2. The next-to-last column gives times for the best version of the TANICCG method in [3], and the last column again shows the ratios of the TANICCG times to those of the new method. (The times for TANICCG given in [3] were obtained on a CRAY 2 at the NASA Langley Resarch Center and do not agree with those of Table 2, which were obtained on a CRAY 2 at the University of Minnesota Army High Performance Computing Research Center.) In [3] we used the convergence test

$$\|F(\mathbf{u}^k)\|_2\leqq\|F(\hat{\mathbf{u}})\|_2, \tag{2.5}$$

where F is the function (1.2), $\mathbf{u}^k$ is the iterate of (1.5), and $\hat{\mathbf{u}}$ is the exact solution of the differential equation evaluated at the grid points. This test, although impractical in practice, ensures that the convergence error is commensurate with the discretization error. A similar test was used for the Newton iterates $u_P^{(k)}$ for (1.10):

$$|\phi(u_P^{(k)})-w_P|\leqq|\phi(\hat{u}_P)-w_P|. \tag{2.6}$$

TABLE 2
Times (seconds) for three-dimensional problem on a CRAY 2.

N	FPS	NL	Total	TANICCG	Speedup
31	.06	.05	.11	1.1	10
63	.31	.35	.66	17.4	25

Here, w_P is the solution of the discrete Poisson problem, and $\hat{u}_p$ is the exact solution of the differential equation. The test (2.6) was implemented as a vector compare across all the grid points so that the vector lengths remain the same for all tests. This may cause some additional work for equations that have already converged, and other tests may be more efficient on some problems. (In our case, no more than three Newton iterations were required for each equation.) Although (2.5) and (2.6) are different tests, each suited to their particular methods, we also used (2.5) for the vector of one-dimensional Newton iterates of the new method and found that the same number of iterations were required as using (2.6). The actual errors in the final iterates produced by the two methods differ by about 15 percent.

The times in Table 2 for the TANICCG method are significantly worse than those of the new method, but some caveats are in order. The ICCG method in [3] was implemented by using the red/black ordering to obtain long vector lengths for the CRAY 2. It is known (see, e.g., [6] and, for a recent review, [7]) that this ordering may seriously degrade the rate of convergence of ICCG, and it is quite likely that better results would be obtained by using the diagonal ordering, as, for example, in [1]. Also, more sophisticated versions of ICCG or the use of other methods might give better results. However, on the basis of the times in Table 2, it is reasonable to conjecture that no iterative method for solving the Newton systems (1.3), or some approximation of them such as (1.5), will be competitive with the new method. Moreover, our fast Poisson solver was a triple Fourier analysis method that used the vectorized one-dimensional fast sine transformations of VFFTPACK obtained from *netlib*. It is known that other fast Poisson solvers that use, for example, cyclic reduction, are faster so that this would give further advantage to the new method.

3. Further discussion of the method. Although the results in the preceding section indicate significant potential for the new method, we must point out several limitations. First, it is necessary to integrate $K(u)$ to obtain $\phi(u)$. Even for the relatively simple function (2.2), the corresponding ϕ of (2.3) is rather complicated and was found only by using MACSYMA. For other K it may be impossible or impractical to obtain the integral explicitly. Recall that ϕ is needed in (2.1) to obtain the boundary values of the Poisson equation and to solve the nonlinear equations (1.10). In both cases, ϕ could be approximated by numerical integration, but this would introduce additional error as well as computing time. Moreover, it seems impossible to apply this approach if K is also an explicit function of the spatial variables, $K = K(u, x, y, z)$, or if K is a vector, as in the equation $(K_1(u)u_x)_x + (K_2(u)u_y)_y = f$.

One reason the times presented in the preceding section were so good is that a FPS could be used. If the domain is such that this is not possible, a multigrid iteration might be a good choice for solving the Poisson equation. If a conjugate gradient method is used, however, then the solution of the Poisson equation may be considerably more expensive. This is illustrated in Table 3, where the discrete Poisson equation of the three-dimensional problem of the preceding section is solved by ICCG. The time for this is given in the column labeled ICCG; NL is, as before, the time to solve the

TABLE 3
Solution of Poisson equation by ICCG *on a CRAY* 2.

N	ICCG	NL	Total	Previous total
31	.527	.046	.57	.11
63	9.87	.38	10.25	.70

nonlinear equations (1.10). The last column in Table 3 reproduces the times from Table 2 using the FPS. Note that even without the FPS, the new method is still almost twice as fast as the TANICCG method.

Clearly, the new method is potentially very good for parallel computation since the solutions of the one-dimensional equations are completely independent. However, the load balancing could be degraded by an unequal number of Newton iterations on different equations. For example, if the initial approximation at grid point P_1 is much better than that at grid point P_2, the Newton iteration at P_2 could take longer to converge. On the other hand, on distributed memory machines, no communication is needed in the Newton iterations except for the convergence test. Most important, if the domain is such that a fast Poisson solver can be used, then a good FPS routine must be available for the particular parallel machine.

Finally, another possible benefit of the new method is discretization error. The best results we have been able to obtain mathematically [2] for the discretization (1.2) of (1.1) is that the local discretization error is $O(h)$, although experimental results indicate it is probably $O(h^2)$. For the new method, however, the only discretization error occurs in the Poisson equation and is $O(h^2)$.

Acknowledgments. We are indebted to Professor Irena Lasiecka of the University of Virginia for reference [4], to Barry F. Smith of Argonne National Laboratory for his assistance in using the VFFTPACK package, and to an anonymous referee for suggestions that improved the presentation.

REFERENCES

[1] C. Ashcraft and R. Grimes, *On vectorizing incomplete factorization and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 122–151.

[2] B. Averick, *Solution of Nonlinear Poisson-Type Equations*, Ph.d. thesis, Dept. of Applied Math., Univ. of Virginia, Charlottesville, VA, January 1991.

[3] B. Averick and J. Ortega, *Solution of nonlinear Poisson-type equations*, Appl. Numer. Math., 8 (1991), pp. 443–455.

[4] M. Crandall, *Semigroups of nonlinear transformations in banach spaces*, in Contributions to Nonlinear Functional Analysis, E. Zarantonello, ed., Academic Press, New York, 1971, pp. 157–179.

[5] R. Dembo, S. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[6] I. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[7] J. Ortega, *Orderings for conjugate gradient preconditionings*, SIAM J. Optimization, 1 (1991), pp. 565–582.

[8] Y. Saad and M. Schultz, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 49–67, January 1993

A DOMAIN DECOMPOSITION METHOD FOR INCOMPRESSIBLE VISCOUS FLOW*

JOHN C. STRIKWERDA† AND CARL D. SCARBNICK‡

Abstract. A method for using domain decomposition to solve the equations of incompressible viscous flow is presented. The method is described in detail, and test results are given for two test problems. A notable feature of the method is that the incompressibility constraint is never explicitly imposed. The domain decomposition uses finite difference and spectral methods on overlapping domains, with second-order accurate interpolation of the velocity relating the solutions on the different domains. The method is shown to be globally second-order accurate by the test results.

Key words. domain decomposition, incompressible viscous flow, overlapping grids

AMS(MOS) subject classifications. 65N20, 65N05

1. Introduction.

In applying finite difference methods to a specific problem, one of the most important considerations is the design of the finite difference grid. The choice of a finite difference grid is easily made for problems with simple geometry, such as rectangular or circular domains, however, for problems on domains which are more complex the designing of a grid can be a significant problem. The choice of grid also affects many aspects of the finite difference method. The efficiency of the numerical solution algorithm and the accuracy of the computed solution are strongly dependent on the finite difference grid. The type of grid can also restrict the choice of the numerical algorithm.

In this paper we present a method of using overlapping grid systems for solving incompressible fluid dynamics problems. By using several grids that overlap, one has more flexibility in the placement of grid points than by using one global grid system. The crucial aspect of using overlapping grids is the interpolation of values between grids. This paper discusses the grids and interpolation methods used on particular test problems, and presents insights into the use of overlapping grids for other problems. It is shown that overlapping grid systems can be used to obtain accurate solutions to the equations for incompressible viscous flow. The solutions are obtained efficiently using iterative methods.

Our method is a special case of the general method of domain decomposition, which has been applied to single second-order elliptic equations by many researchers (see the collection of papers in [5]), and to incompressible viscous flow in [1], [3], and [4] using the finite element method. Much of the work on domain decomposition revolves around methods of accelerating the iterative solution procedure and implementation of domain decomposition on parallel computers. The emphasis of this paper is on the features of the domain decomposition that are independent of the particular solution algorithm used on each domain. The iterative method used on the subdomains in this work is not to be regarded as very fast. In a subsequent publication we will describe an iterative method that, based on simple test cases, promises to significantly improve the convergence rate [12].

*Received by the editors December 4, 1989; accepted for publication (in revised form) November 20, 1991.

†Department of Computer Sciences and the Center for Mathematical Sciences, University of Wisconsin, Madison, Wisconsin 53706. This research was partially supported by Naval Sea Systems Command contract N00024-85-M-B720 and U.S. Army Research Office contract DAAL03-87-K0028.

‡San Diego Supercomputer Center, P.O. Box 85608, San Diego, California 92186-9784.

A notable feature of the method presented here is the handling of the conservation of mass. By properly reformulating the equations, the troublesome integrability constraint on the boundary data is removed. The divergence-free nature of the solution is obtained as a result of the consistency of the scheme to the differential equation.

In this paper we regard the overlapping grids as several overlapping domains, each with its own grid which is independent of other grids. Thus we refer to our method as the method of overlapping domains. This way of referring to the method calls attention to the central problem of relating the values of the solution on the several domains and deemphasizes the construction of the particular grids. As shown in §5, some of the primary difficulties are not specific to the choice of grid or the finite difference scheme.

We also restrict our attention to the steady Stokes equations because they are useful to illustrate the basic ideas of the method of overlapping domains as applied to incompressible flow. Overlapping domains can be used with the incompressible Navier–Stokes equations, both steady and time-dependent. The extra features introduced by the more general equations can be incorporated into the method without significant difficulties. The method is applicable to three-dimensional problems.

Also, related to the work presented here is the work of Thompson and Ferziger [17] and Vanka [18], in which the multigrid iteration scheme was applied to the incompressible Navier–Stokes equations. Each of these works used staggered grids which limit their methods to rectilinear domains. As we demonstrate by our computational results, our method maintains second-order accuracy even when the grids are not both rectilinear (see also [10]). A use of the multigrid iteration method to accelerate the rate of convergence of the iterative method will appear soon [12].

A typical example of the use of overlapping grids is in the computation of the flow past several bodies with nonsimple shapes. Boundary-fitted coordinate systems should be used near the bodies to insure good accuracy. However, away from the bodies it is simplest and most convenient to use a standard global cartesian grid or other such simple grid. One may indeed blend the boundary-fitted grids with the grid system used away from the bodies, but it is not done easily, nor can it be done by any simple general procedure. The approach advocated here is that of constructing the boundary-fitted and the global grids independently, but large enough to overlap. The values of the solution on the boundary of each grid are determined by interpolation of values from the other grids.

The chief advantages of the use of overlapping grids are the flexibility and generality of the method. It can be used with any geometric configuration and, in principle, the grid can be altered as needed. Local refinement grids and adaptive grid methods can also be used to improve the resolution of the solution. The flexibility of the method can be seen in comparison with global grid generation techniques and grid patching techniques. The need for higher resolution in one area of the domain usually forces both higher resolution in areas that do not need the resolution and lower resolution in other areas.

The use of overlapping grids can be used with the grid generation method to produce grids which enable solutions to be computed efficiently. The elliptic grid generation method of Thompson [16], and methods of Kreiss [7] and Chesshire [2], can be used to construct boundary-fitted grids around bodies. Farther from the bodies a cartesian grid or similar simple grid can be used. Boundary-fitted grids are essential for obtaining accurate solutions near bodies, while the use of simple cartesian grids is important for obtaining efficiency in computations.

2. The Stokes equations on overlapping domains. We first consider questions of uniqueness and existence of solutions to the steady Stokes system of equations on overlapping domains. These questions are not difficult to resolve, but they are interesting and important because of the insight they give to the numerical approximations.

We begin by considering two domains Ω_1 and Ω_2 which overlap, i.e., $\Omega_1 \cap \Omega_2$ is nonempty (see Fig. 1). We also assume that neither domain is included in the other. We consider the Stokes system of equations, given by

$$\nabla^2 \vec{u} - \vec{\nabla} p = 0, \tag{2.1}$$

$$\vec{\nabla} \cdot \vec{u} = 0, \tag{2.2}$$

on each separate domain. Boundary data for each domain is specified only on that part of the boundary which is not interior to the other domain. For purposes of discussion, we assume that the velocity $\vec{u}$ is specified by data $\vec{b}$ on the boundary of $\Omega_1 \cup \Omega_2$. For that part of $\partial\Omega_1$ (the boundary of Ω_1) that lies within Ω_2 we require that the velocity be equal to the velocity obtained from the solution of the Stokes system on Ω_2, and similarly for that portion of $\partial\Omega_2$ within Ω_1.

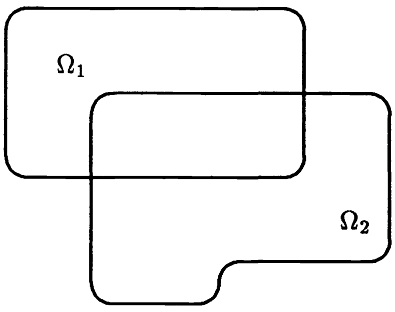


FIG. 1

We also assume that the integrability condition holds on the whole domain, i.e.,

$$\int_{\partial(\Omega_1 \cup \Omega_2)} \vec{b} \cdot \vec{n} = 0. \tag{2.3}$$

In our finite difference method some of the boundary data for a subdomain will be obtained by interpolation from other domains, and it would be difficult to explicitly impose the integrability constraint on the data for each subdomain. Thus we will consider a slightly more general case in which the divergence of the velocity field is required to be constant. That is, (2.2) is replaced with

$$\vec{\nabla} \cdot \vec{u}_i = d_i \quad \text{on} \quad \Omega_i. \tag{2.4}$$

The constant d_i is determined by the integrability condition for the domain, i.e.,

$$\int_{\partial\Omega_i} \vec{u}_i \cdot \vec{n} = d_i \; |\Omega_i|. \tag{2.5}$$

As will be seen, this formulation avoids the difficulties inherent in methods that require that the boundary data satisfy the integrability constraint on each subdomain; see [1].

The mathematical statement of the problem is then described as follows. In Ω_1,

$$\nabla^2\vec{u}_1 - \vec{\nabla}p_1 = 0, \qquad \vec{\nabla}\cdot\vec{u}_1 = d_1, \tag{2.6}$$

and in Ω_2,

$$\nabla^2\vec{u}_2 - \vec{\nabla}p_2 = 0, \qquad \nabla\cdot\vec{u}_2 = d_2 \tag{2.7}$$

with boundary conditions

$$\begin{aligned} \vec{u}_1 &= \vec{b} \quad \text{on} \quad \partial\Omega_1\backslash\Omega_2, \\ \vec{u}_1 &= \vec{u}_2 \quad \text{on} \quad \partial\Omega_1\cap\Omega_2, \\ \vec{u}_2 &= \vec{b} \quad \text{on} \quad \partial\Omega_2\backslash\Omega_1, \\ \vec{u}_2 &= \vec{u}_1 \quad \text{on} \quad \partial\Omega_2\cap\Omega_1, \end{aligned} \tag{2.8}$$

where $\vec{b}$ is the velocity data on the boundary of $\Omega_1 \cup \Omega_2$.

We make the assumption that the Stokes system of equations (2.6), (2.7) has a unique solution for each domain here when the velocity is specified on the boundary. By a unique solution we mean that the velocity function is determined uniquely and that the pressure function is determined to within an additive constant. The uniqueness of the solution to (2.6), (2.7) with the velocity specified on the boundary follows from the uniqueness of the Stokes system (2.1), (2.2). This assumption is certainly justified for all problems of physical significance. See Temam [15] for a discussion of existence and uniqueness of the stationary Stokes equations.

Notice that we do not assume a priori that $\vec{u}_1$ and $\vec{u}_2$ are the same on $\Omega_1\cap\Omega_2$. Also, notice that the pressure on one domain does not directly interact with the pressure on the other domain. In particular, we cannot specify the pressure as a boundary condition along with the velocity, as this would result in overdetermined boundary value problems on the subdomains.

It is useful to consider the following iterative procedure, essentially the Schwarz alternating procedure (see [8]), to solve the overlapping domain problem. We begin with any velocity field $\vec{u}_1^0$ defined on Ω_1. This need not be divergence-free. Using $\vec{u}_1^0$ as the boundary condition on $(\partial\Omega_2)\cap\Omega_1$, solve the system (2.7) for $\left(\vec{u}_2^0, p_2^0\right)$. Then using $\vec{u}_2^0$ as boundary data on $(\partial\Omega_1)\cap\Omega_2$, solve (2.6) for $\left(\vec{u}_1^1, p_1^1\right)$. Continue in this way, so that $(\vec{u}_1^\nu, p_1^\nu)$ is determined from $\vec{u}_2^{\nu-1}$ and $(\vec{u}_2^\nu, p_2^\nu)$ is determined from $\vec{u}_1^\nu$. The values of d_1^ν and d_2^ν are determined at each step by the integrability conditions. Supposing that the iterative method converges, we may indeed ask if the final values of constants d_1 and d_2 are the same, and if $(\vec{u}_1, p_1)$ and $(\vec{u}_2, p_2)$ together may be regarded as a solution on the total domain $\Omega_1\cup\Omega_2$. The convergence of this procedure can be shown using the methods of [8].

First, a solution to this problem exists. This is seen by considering the one problem defined on $\Omega_1 \cup \Omega_2$, the union of the two domains. The data vector function $\vec{b}$ is defined on the boundary of $\Omega_1\cup\Omega_2$, and thus by our assumption, a solution exists.

Next we show that the solution is unique. Assuming that solutions $(\vec{u}_1, p_1)$ and $(\vec{u}_2, p_2)$ exist on Ω_1 and Ω_2, respectively, the first question to be addressed is whether they agree on the intersection $\Omega_1 \cap \Omega_2$. Since $\vec{u}_1$ and $\vec{u}_2$ are equal on the boundary of $\Omega_1 \cap \Omega_2$, it follows by our uniqueness assumption that there is a unique solution and that d_1 and d_2 must be equal by the integrability condition on $\Omega_1 \cap \Omega_2$. Moreover, the pressure functions p_1 and p_2 differ at most by an additive constant. It then follows that the two solutions $(\vec{u}_1, p_1)$ and $(\vec{u}_2, p_2)$ can be regarded as a solution to the global boundary value problem on $\Omega_1 \cup \Omega_2$. Hence d_1 and d_2 are zero because of the integrability condition (2.3) on $\Omega_1 \cup \Omega_2$.

An observation that is important in the numerical approximation is that the two pressure fields, p_1 and p_2, differ by a constant whose value is not determined by the solution. Thus there are two undetermined parameters in the solution: the average value of p_1 on Ω_1 and the average value of p_2 on Ω_2. In general, there is an undetermined additive constant for the pressure on each subdomain in the decomposition.

It is important to note that it is by comparing the two different problems on the intersection of the domains that we see that they define a global solution on the union of the domains. This suggests that the *conditioning* of the problem is dependent on the size of the intersection. This conditioning is evident in the numerical investigations. A smaller overlap domain requires more iterations for convergence and gives less accuracy in the answers, although the same order of accuracy.

The increase in error with the smaller overlap is most likely due to the greater interaction between the interpolation done on the two boundaries. The error induced by interpolation to a boundary point on domain 1 affects the solution in the interior of domain 1. The use of interpolation from domain 1 back to the boundary of domain 2 carries that error to domain 2. Thus the errors introduced by interpolation are coupled together. However, because the system is elliptic, the effect of boundary errors on the solution at a point is smaller the farther the point is from the boundary; see [14]. Thus a larger overlap region could be expected to reduce the coupling between the interpolations at the two boundaries.

3. Interpolation between domains.

An important feature of the approach taken here is that the overall method is composed of relatively independent components. For example, the finite difference methods used on the different subdomains need not be the same. Also, the exact form of the iterative methods on the subdomains do not depend on each other.

The topic of this section, namely, the interpolation between domains, is also relatively independent of the other aspects of the overall problem. In particular, the interpolation is not dependent on the finite difference methods nor on the iterative method. As shown in this section, the choice of interpolation method is primarily based on convenience and the need for accuracy in the solution.

For each of the test cases for which the numerical experiments were made there were two grids. In the first set of cases the two grids are both cartesian grids, but rotated with respect to each other (see Fig. 2), while for the second set of test cases, one grid is a standard cartesian grid with uniform spacing and the other grid is a polar coordinate grid, also with uniform spacing in each coordinate direction (see Fig. 3). Rather than discuss the interpolation in general, we will confine ourselves to these two situations.

The basic interpolation problem is a local problem, that is, given a discrete function defined on a cartesian grid, how should a value be assigned to a point that is not on the grid? To mathematically formulate the problem in two space dimensions we consider a uniform cartesian grid with points (x_i, y_j) given by $x_i = ih$ and $y_j = jh$, where the grid spacing h is a positive number. Given a general point $(\bar{x}, \bar{y})$ there is a grid point (x_i^*, y_j^*) which is closest to $(\bar{x}, \bar{y})$. A second parameter of significance is the quadrant relative to (x_i^*, y_j^*) in which $(\bar{x}, \bar{y})$ lies. Without loss of generality we

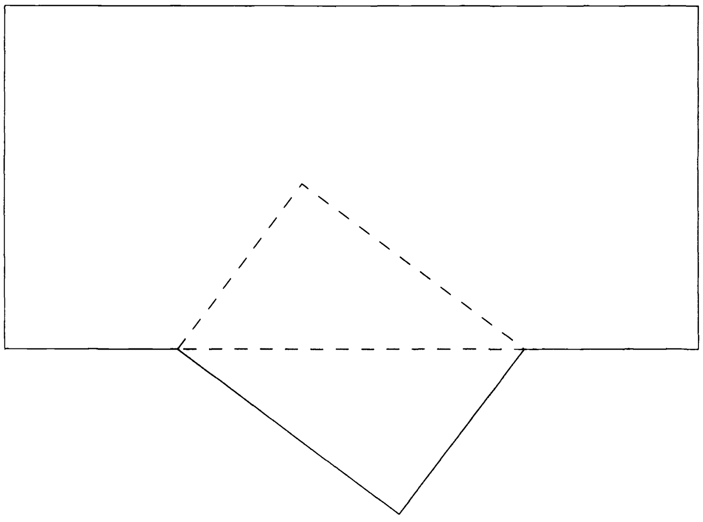


FIG. 2

can assume that (x_i^*, y_j^*) is $(0, 0)$. The general case can be reduced to this special case by a simple translation. We can also assume, to simplify the discussion, that $(\bar{x}, \bar{y})$ is in the first quadrant, that is, both $\bar{x}$ and $\bar{y}$ are positive.

The first interpolation method that we consider is bilinear interpolation. In this case the value of a function at $(\bar{x}, \bar{y})$ is given by

$$\begin{aligned} f &= f_{0,0} + \bar{x}\left(f_{1,0} - f_{0,0}\right)/h + \bar{y}\left(f_{0,1} - f_{0,0}\right)/h \\ &\quad + \bar{x}\bar{y}\left(f_{1,1} - f_{1,0} - f_{0,1} + f_{0,0}\right)/h^2, \end{aligned} \tag{3.1}$$

where $f_{0,0} = f(x^*, y^*)$, $f_{1,0} = f(x^* + h, y^*)$, $f_{0,1} = f(x^*, y^* + h)$, and $f_{1,1} = f(x^* + h, y^* + h)$. Bilinear interpolation is only accurate of order one. The order of accuracy is equal to the highest degree of polynomial for which the interpolation must be exact. Thus (3.1) is exact for any polynomial of first degree, but is not exact for all second-degree polynomials. The use of bilinear interpolation gave only first-order accuracy in the solution.

The second interpolation method that we consider is quadratic interpolation (see Fig. 3). The particular formula we used is

$$\begin{aligned} f &= f_{0,0} + \bar{x}\left(f_{1,0} - f_{-1,0}\right)/2h + \bar{x}^2\left(f_{1,0} - 2f_{0,0} + f_{-1,0}\right)/2h^2 \\ &\quad + \bar{y}\left(f_{0,1} - f_{0,-1}\right)/2h + \bar{y}^2\left(f_{0,1} - 2f_{0,0} + f_{0,-1}\right)/2h^2 \\ &\quad + \bar{x}\bar{y}\left(f_{1,1} - f_{1,0} - f_{0,1} + f_{0,0}\right)/h^2. \end{aligned} \tag{3.2}$$

In the test cases it was found that quadratic interpolation gave overall second-order accuracy, which is the same order as the finite difference schemes, so no methods of higher order were investigated.

Henshaw and Chesshire [6] reduced the amount of overlap as the grid spacing was reduced, and reported that a third-order interpolation was necessary to maintain the overall second-order accuracy of the solution. Since we consider the domain fixed, independent of the grid spacing, we are able to use interpolation of the same order as the scheme.

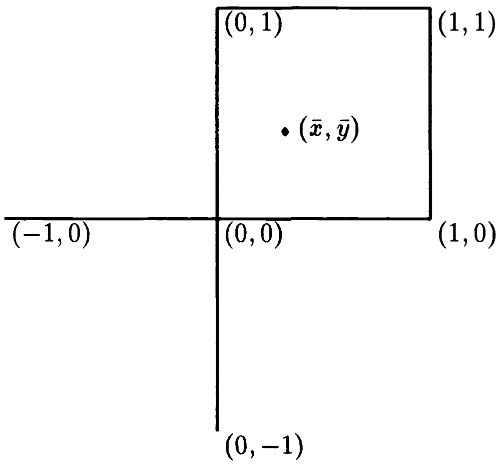


FIG. 3

For the interpolation from the polar grid to the cartesian grid in the second test problem, a method based on Fourier interpolation was used. This method was very natural to use because the numerical method on the polar grid used finite Fourier series to approximate the derivatives with respect to the angular variable. The finite Fourier series are discussed in §6.

The polar grid used in the numerical experiments consisted of grid points $(r_i,\ \theta_j)$ where $r_i = r_{\min} + i \cdot \Delta r$ for $0 \le i \le N$ and $\theta_j = j \cdot \Delta\theta$ for $0 \le j \le J$ with $\Delta\theta = \pi/J$ and $\Delta r = (r_{\max} - r_{\min})/N$. The even spacing of the grid is not crucial to the method; it was used for simplicity. Indeed, because the grid spacing in the two grids was independent of each other, good resolution with the polar grid could easily be obtained using a uniform grid without significantly increasing the overall computational effort.

The actual interpolation procedure was as follows. Given a point $(\bar{x},\bar{y})$ on the boundary of the cartesian grid for which data had to be supplied from the polar grid for a function f, its polar coordinates $(\bar{r},\bar{\theta})$ were first computed. The three values of r_i nearest to $\bar{r}$ were determined, and then using Fourier interpolation, values of $f(r_i,\bar{\theta})$ were obtained. Finally, one-dimensional quadratic interpolation in r was used to obtain $f(\bar{r},\bar{\theta})$.

Both the bilinear interpolation (3.1) and quadratic interpolation (3.2) were tested in place of this method to interpolate values from the polar grid to the cartesian grid. Both of these methods were far less accurate than was the use of the finite Fourier series. The inaccuracy of the quadratic interpolation for this case is due to the large grid spacing in the angular direction.

To interpolate the velocity from one grid to another, the different representation of the velocity on each grid must be accounted for. We found it best to interpolate the velocity components independently, and then transform them to the proper representation. For example, in interpolating from a cartesian representation to a polar representation, the cartesian velocity components are interpolated, and then transformed to the polar representation of the velocity.

The basic data for interpolation to a given point from a grid consists of the coordinates of the nearest grid point and the quadrant in which the given point lies relative to the nearest grid point. In many applications it would be best to determine this data prior to the main calculation. In the applications employed here the basic data was computed as needed. Because of the simplicity of the two grids used, i.e., cartesian and polar, the necessary computations were relatively easy and straightforward. For calculations being done on vector processors it would presumably be more efficient to store the interpolation data, rather than recompute it. This would also be true when using any grid for which it is difficult to determine the nearest grid point to a given point.

4. The iterative solution procedure. Before discussing the finite difference approximation we give a brief description of the iterative solution procedure. This is done before the discussion of the finite difference methods themselves to highlight those aspects which relate to the considerations discussed in the section on the overlapping domains for the Stokes system.

The (inner) iterative method used on the cartesian domain is based on point successive overrelaxation and is described in detail in [11]. The iterative method used on the polar domain is based on line successive overrelaxation and is well suited for methods using finite Fourier series methods. This method was also used in [13], where it is described in detail. Multigrid methods are also suitable; see [12]. One outer iteration consists of a single inner iteration on each subdomain with boundary data obtained from the other domain.

The basic inner iteration step to update the solution on a domain consists of two parts. The first part consists of updating the velocity using one step of a method, such as successive overrelaxation. This step uses only the first vector equation in (2.6) or (2.7). In the second part, the pressure is updated based on the local velocity divergence, which is the second equation in (2.6) or (2.7).

For the cartesian grid, assuming equal grid spacing of h and assuming the standard second-order accurate five-point Laplacian, these formulas are:

$$\begin{aligned} u_{ij} &\leftarrow u_{ij} + \omega\left(\frac{1}{4}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) - u_{i,j} - \frac{h^2}{4}\delta_{x,r}p_{i,j}\right), \\ v_{ij} &\leftarrow v_{ij} + \omega\left(\frac{1}{4}(v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1}) - v_{i,j} - \frac{h^2}{4}\delta_{y,r}p_{i,j}\right), \end{aligned} \tag{4.1}$$

where $\delta_{x,r}$ and $\delta_{y,r}$ represent regularized central difference approximations to the derivatives with respect to x and y, respectively. Formulas for $\delta_{x,r}$ and $\delta_{y,r}$ are given in §6. The algorithm uses immediate replacement, so that only one copy of the solution needs to be kept. The iteration parameter ω was chosen as for standard successive overrelaxation (see §7).

After the velocity has been updated, the pressure is updated by

$$p_{ij} \leftarrow p_{ij} - \gamma(\delta_{x,r}u_{ij} + \delta_{y,r}v_{ij}) \tag{4.2}$$

for all interior points. Here, as before, $\delta_{x,r}$ and $\delta_{y,r}$ represent regularized central difference approximations, but have a different shift than they did for the gradient terms in (4.1). The iteration parameter γ was chosen proportional to the grid spacing; see §7 for more on the choice of γ. After all the interior points were updated, the boundary values of the pressure are set by quadratic extrapolation using formulas such as

$$p_{0j} = 3(p_{1j} - p_{2j}) + p_{3j}$$

along the boundary given by $i = 0$. Similar formulas are used along the other boundaries. This extrapolation is of high enough order not to affect the overall second-order accuracy of the solution (see [14]). The stopping criteria for the iterative method are discussed in §7. The formula to update the pressure, given by (4.2) is part of the classical Uzawa algorithm and related methods (see [15, p. 139]).

When solving a problem on overlapping domains the procedure we have adopted is as follows. The solution on one domain, say Ω_1, is updated as in one step of the iterative method for a single domain. The values of the velocity on the boundary of Ω_2 are determined by interpolation from the latest values on Ω_1, and then the solution is updated on Ω_2 as in one step of the iterative method for a single domain. The boundary values of Ω_1 are then updated by interpolation of the velocity on Ω_2. This describes one step of the outer iterative method. The iteration proceeds until the solution on each domain has converged to within the specified tolerance.

Note that since only one sweep of the inner iteration is done on each domain, this method is *not* the Schwarz method. The Schwarz method requires that the solution be obtained on each subdomain before moving to the next domain.

Convergence is determined by examining the norm of the changes in velocity and the deviation of the changes of the pressure from a constant (see §7). Thus, the solution is determined as converged when on each subdomain the changes in velocity are small and the changes in the pressure are essentially constant.

Because the pressure and changes in pressure are determined only to within an additive constant, they are measured in the norm of L^2 *modulo constants*. By (4.2) the convergence requirement on pressure is equivalent to requiring that the discrete divergence of the velocity field is also constant, i.e., it is d_i on subdomain Ω_i. The relationship between the change in pressure and the value of d_i is that the change in the pressure on Ω_i is $-\gamma d_i$. Since the constants d_i are set by the boundary data and the interpolation between domains, this shows that one cannot use formula (4.2) and expect that the pressure changes can converge to zero. Note that the values of d_i are not needed explicitly because of the choice of norms. West's algorithm [19] is used to compute the norms of the pressure and the pressure changes. The mean values of the pressure and pressure changes are computed during the computation of the norms, but are not needed elsewhere in the calculation.

It is important to notice that at no time is it required that the velocity field be divergence-free or satisfy a similar numerical condition. Also, the pressure is not constrained so as to determine the unimportant additive constant. This avoids entirely problems such as those arising in the use of divergence-free finite elements (see [1]).

In particular, (4.2) does not enforce the condition that the discrete divergence must be zero at convergence, it only enforces the condition that the discrete divergence is constant on each subdomain. This is similar to the algorithm discussed in §2, and as in the proof of the uniqueness of the solution in §2, the divergence-free property is a result of the properties of the system and the boundary data.

Notice that the pressure fields on the two domains do not directly affect each other. In particular, their difference in the overlap is not restricted to being a constant. In fact, it is not clear how best to compare the pressure values, since they are defined on different grids.

These observations can be used to give a posteriori error checks. The deviation of the velocity fields from being divergence-free and the deviation from a constant of the difference of the two pressure fields on the overlap can be used to indicate the accuracy of the final solution.

We have made no attempt to prove that this iterative method will converge. It appears that such a proof is beyond our current methods of analysis. In practice, the method has performed very well. The solutions have been obtained in a reasonable number of steps.

For the second test problem, the size of the overlap region had a noticeable effect on the results. The method converged faster and gave more accurate results, when the overlap region was larger. One result of a smaller overlap region is that the errors resulting from interpolation from one domain to another have a greater interact with each other. Elliptic systems of finite difference schemes have the property that non-smooth boundary data give rise to smooth solutions in the interior of domains [14]. The effect of irregularities in boundary data obtained from interpolation from another domain will be smoothed out in the interior of the domains, but if the data used for interpolation is taken from near the boundary, some effect of the boundary irregularity can be propagated back to the other domain. The size of the overlap region affected only the magnitude of the errors, not the actual order of accuracy.

5. The reentrant corner difficulty. One difficulty which was discovered and finally surmounted is of sufficient interest to warrant special attention. This difficulty occurred with the second test problem and has to do with the corners of the cartesian grid resulting from the exclusion of portions of that grid from the computational domain. These corners, such as the points R_1 and R_2 in Fig. 4, are distinguished by being reentrant corners.

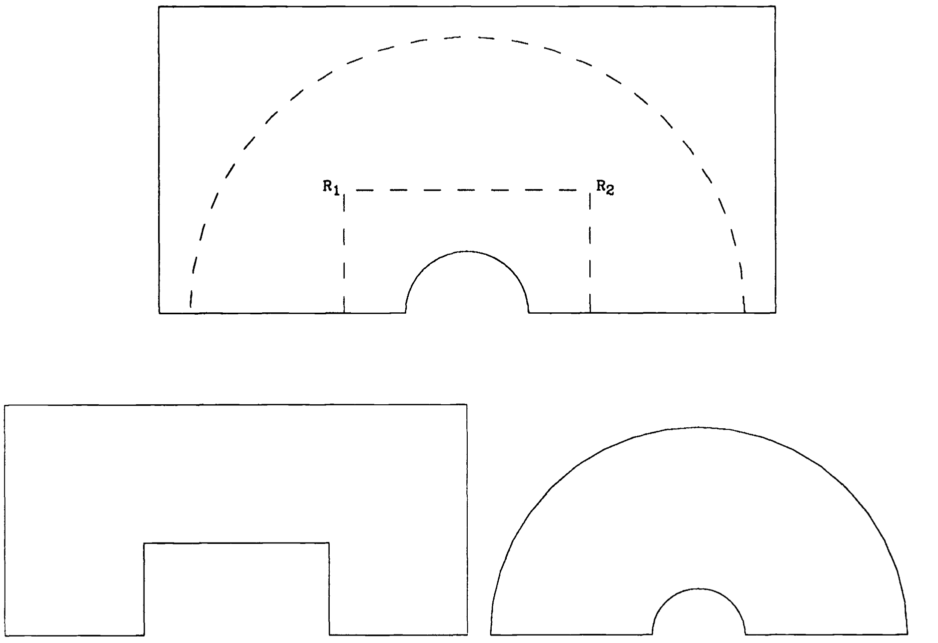


FIG. 4

For the Navier–Stokes or Stokes equations in a region with reentrant corners the solution will in general have a pressure singularity at such a corner. Usually this is studied only for flows over steps, in which the velocity is zero along the sides of the step (see [9]), however, the pressure singularity exists at reentrant corners for general data.

For the overlapping domains of the second problem the true solution has no singularity at the reentrant corners since the reentrant corners are interior to the other domain. However in the numerical approximation, the data given by the interpolation introduces errors which cause a pressure singularity to appear in the error. In the test problem the graphical display of the error showed that these pressure singularities were the dominant feature of the error. The singularity in the pressure error caused a vortical flow in the velocity error. The discovery of the pressure error singularity was made possible by graphically displaying the errors.

Two approaches were used to improve the solution by decreasing the strength of these pressure error singularities. The first of these we call "rounding off the corner." It is based on the observation that the grid points at the reentrant corners can be treated as interior points of the cartesian grid. Both the pressure and velocity at these corner points were updated as interior points. This is possible because all the nearest neighbors are also part of the grid. Interpolation is done then only at points which are missing at least one neighbor in the grid. This simple procedure gave a dramatic improvement in the accuracy of the solution.

The second method was to "chop off" the corner so that the one 270° angle is replaced by two 225° angles. This method introduced more programming difficulties and did not give a significant improvement over the method of rounding off the corner, and so will not be discussed further.

It is important to remember that the reentrant corner pressure error singularity is inherent in the use of overlapping domains. This can be seen by considering the overlapping domain problem for the differential equations as discussed in §2. If we assume that the specification of boundary values on the overlap portions of the boundary, i.e., $(\partial\Omega_1)\cap\Omega_2$ and $(\partial\Omega_2)\cap\Omega_1$, is not exact but contains some error, then in general there will be a pressure singularity at each reentrant corner. The strength of the singularity will be directly related to the amount of error in the boundary specification.

6. The finite difference method for the Stokes equations. The finite difference methods used in the numerical experiments with the overlapping domain method are variations of the regularized difference schemes introduced by the first author [10]. This class of schemes is the only one for which published results are given to demonstrate the second-order accuracy of the solution in nonsimple domains.

In cartesian coordinates the Stokes equations are

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}-\frac{\partial p}{\partial x}=0, \tag{6.1}$$

$$\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}-\frac{\partial p}{\partial y}=0, \tag{6.2}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0, \tag{6.3}$$

and in polar coordinates they are

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}-\frac{u}{r^2}+\frac{2}{r^2}\frac{\partial v}{\partial \theta}-\frac{\partial p}{\partial r}=0, \tag{6.4}$$

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial v}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2 v}{\partial \theta^2}-\frac{v}{r^2}-\frac{2}{r^2}\frac{\partial u}{\partial \theta}-\frac{1}{r}\frac{\partial p}{\partial \theta}=0, \tag{6.5}$$

$$\frac{1}{r}\frac{\partial (ru)}{\partial r}+\frac{1}{r}\frac{\partial v}{\partial \theta}=0. \tag{6.6}$$

In (6.1), (6.2), and (6.3) the variables (u,v) denote the velocity components in the cartesian representation, while in (6.4), (6.5), and (6.6) they denote the components in the polar representation. No confusion should result from not distinguishing between the choice of representation since the two coordinate systems are treated separately.

The first test problem is on the domain consisting of the rectangle $-1 \le x \le 1$, $0 \le y \le 1$, with a right triangle with legs of length 0.8 and 0.6 attached to the rectangle along the hypotenuse at $-0.5 \le x \le 0.5$ (see Fig. 2). The two grids used to compute the solution were both cartesian grids. On each domain the equations have the form (6.1)–(6.3), but the velocity components on one grid must be rotated by the angle of the tilt between the grids to obtain the velocity components on the other grid.

The region for which the second set of numerical experiments were run is the rectangle $-1 \le x \le 1, 0 \le y \le 1$ with the exclusion of a circle of radius r_{min} centered at the origin. This region is displayed in Fig. 4. The grid system consisted of an evenly spaced cartesian grid with equal spacing in each direction and a polar grid with uniform spacing in each of the radial and angular directions.

The domain covered by the cartesian grid was the large rectangle with the exclusion of a smaller rectangle around the excluded circle as displayed in Fig. 4. The grid spacings on this, the cartesian domain, were given by

$$\Delta x = \Delta y = 1/M \tag{6.7}$$

for some positive integer M. The omitted grid points were those in the rectangle

$$-i_0\Delta x < x < i_0\Delta x, \qquad 0 \le y < i_0\Delta y. \tag{6.8}$$

The domain covered by the polar grid was the region displayed in the lower right portion of Fig. 4, given by

$$r_{\text{min}} \le r \le r_{\text{max}}, \qquad 0 \le \theta \le \pi. \tag{6.9}$$

Various values of r_{min} and r_{max} were used in the studies. Typical values were 0.2 for r_{min} and 0.9 for r_{max}. The grid spacings for this domain, the polar domain, were given by

$$\Delta r = (r_{\text{max}} - r_{\text{min}})/N \quad \text{and} \quad \Delta\theta = \pi/J \tag{6.10}$$

for some positive integers N and J.

In all the tests an analytical solution of the Stokes equations was used to assess the accuracy of the method. The first problem used Dirichlet boundary conditions, in which the velocity was specified along the boundary.

For the second problem the solutions which were used were symmetric about the x axis. The boundary conditions imposed were that the velocity was specified along the boundaries given by $x = -1, x = 1, y = 1$, and also $r = r_{\min}$. Along the boundary $y = 0$ symmetry conditions were used. These were

$$\frac{\partial u}{\partial y} = 0 \quad \text{and} \quad v = 0.$$

For the cartesian grid the finite difference method was based on second-order accurate central differences for all derivatives. However, the use of the standard central difference for the divergence equation and for the pressure gradient results in nonsmooth solutions. The loss of smoothness can be avoided by using regularized central differences, as introduced in [10] (see also [11]). These approximations are

$$\frac{\partial p}{\partial x} \simeq \delta_{x,r}p = \frac{p_{i+1,j} - p_{i-1,j}}{2\Delta x} - \frac{p_{i+2,j} - 3p_{i+1,j} + 3p_{i,j} - p_{i-1,j}}{6\Delta x}, \tag{6.11}$$

$$\frac{\partial p}{\partial y} \simeq \delta_{y,r}p = \frac{p_{i,j+1} - p_{i,j-1}}{2\Delta y} - \frac{p_{i,j+2} - 3p_{i,j+1} + 3p_{i,j} - p_{i,j-1}}{6\Delta y}, \tag{6.12}$$

$$\frac{\partial u}{\partial x} \simeq \delta_{x,r}u = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} - \frac{u_{i+1,j} - 3u_{i,j} + 3u_{i-1,j} - u_{i-2,j}}{6\Delta x}, \tag{6.13}$$

$$\frac{\partial v}{\partial y} \simeq \delta_{y,r}v = \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} - \frac{v_{i,j+1} - 3v_{i,j} + 3v_{i,j-1} - v_{i,j-2}}{6\Delta y}. \tag{6.14}$$

Notice that the additional regularizing terms are third-order divided differences which are shifted forward for the pressure and shifted backward for the velocity derivatives. At grid points near the boundaries, where the approximations (6.11)–(6.14) cannot be applied, the third-order difference is shifted the other way.

For the polar grid of the second problem, a method was employed which uses both finite differences and Fourier methods. The solutions were assumed to be symmetric about the line $y = 0$, and thus the components of the velocity in polar representation and the pressure can be expressed as

$$u_{ij} = u(r_i, \theta_j) = \frac{1}{2}a_0(r_i) + \sum_{k=1}^{J-1} a_k(r_i) \cos k\theta_j + \frac{1}{2}(-1)^j a_J(r_i), \tag{6.15}$$

$$v_{ij} = v(r_i, \theta_j) = \sum_{k=1}^{J-1} b_k(r_i) \sin k\theta_j, \tag{6.16}$$

$$p_{ij} = p(r_i, \theta_j) = \frac{1}{2}c_0(r_i) + \sum_{k=1}^{J-1} c_k(r_i) \cos k\theta_j + \frac{1}{2}(-1)^j c_J(r_i). \tag{6.17}$$

In these formulas $r_i = r_{\min} + i \cdot \Delta r$ and $\theta_j = j\Delta\theta$. The formulas (6.15), (6.16), and (6.17) express the function in terms of their finite Fourier series for a fixed value of r_i. The determination of the coefficients a_k, b_k, and c_k are easily determined from the function values. Because of the assumed symmetry in the solution, u and p are even functions in θ and v is an odd function.

Approximations to the derivatives with respect to θ are obtained by regarding the variable θ as a continuously varying variable on the right-hand side of (6.15), (6.16), and (6.17). For example,

$$\frac{\partial v}{\partial \theta}(r_i, \theta_j) \simeq \sum_{k=1}^{J-1} b_k(r_i) k \cos k\theta_j. \tag{6.18}$$

The finite Fourier representation is used only to obtain the approximations to the derivatives with respect to θ. The derivatives are used to evaluate the left-hand side of equations (6.4), (6.5), and (6.6) as part of the iterative solution technique.

The importance of using regularized differences also applies to the Fourier approximation. In this case, the coefficients $a_J(r_i)$ and $c_J(r_i)$ must be constrained (see also [13]). These coefficients are related through (6.4), but some relation other than given by (6.4), (6.5), and (6.6) is needed to determine these values. To control the magnitude of the coefficients $c_J(r_i)$ in (6.17), the approximation of some derivative must be altered. Note that the derivative of p with respect to θ occurs only in (6.5), which relates the coefficients of p, i.e., the $c_k(r_i)$, with those of v, i.e., the $b_k(r_i)$, but there is no mode for k equal to J in the representation for v. It would be possible to allow for a coefficient $b_J(r_i)$ in the representation for v. This coefficient $b_J(r_i)$ would be proportional to $c_J(r_i)$ and could be constrained through (6.6). Note that since $\sin J\theta_j$ is identically zero, this coefficient would have no direct effect on v. Equivalently, $b_J(r_i)$ was not explicitly used; instead a term equal to

$$-\tfrac{1}{4}(-1)^j c_J(r_i)$$

was added to the approximation of the divergence equation (6.6). The form of the iterative method then forced the coefficient $c_J(r_i)$ to decrease. The rate of decrease is $1 - \gamma/2$, where γ is the iterative parameter in (4.2). Some such constraint on the coefficients $c_J(r_i)$ is necessary; without it, the dominant pressure errors are in the coefficient $c_J(r_i)$. Since this coefficient relates to the highest frequency supported by the grid there is no concern with consistency in treating this term.

7. The numerical experiments.

The first set of numerical experiments were run using the domain shown in Fig. 2. On each domain the grid is a cartesian grid. For the cases shown in Table 1, the grid spacing was $1/M$ in both directions for the larger grid and was $.8/M$ for Δx and $.6/M$ for Δy on the smaller tilted grid. On each domain the Stokes equations have the form (6.1)–(6.3), but the components of the velocity vectors are related by the rotation of the coordinate system. After interpolating the velocity vector components from one grid to the other, the components in the new coordinate system have to be computed using the appropriate rotation.

The solution for which most of the tests of the first problem were made is

$$u = \tfrac{1}{2}\left((x+0.5)^2 - (y+0.5)^2\right) / \left((x+0.5)^2 + (y+0.5)^2\right)$$
$$-\tfrac{1}{2}\log\left((x+0.5)^2 + (y+0.5)^2\right),$$
$$v = (x+0.5)(y+0.5) / \left((x+0.5)^2 + (y+0.5)^2\right),$$
$$p = 2(x+0.5) / \left((x+0.5)^2 + (y+0.5)^2\right) + p_0,$$

TABLE 1

	Main grid errors			
M	u	v	p	d
10	5.43(−4)	4.29(−4)	1.64(−2)	1.7(−3)
16	1.88(−4)	1.58(−4)	4.97(−3)	2.7(−4)
24	6.03(−5)	5.26(−5)	1.69(−3)	6.5(−5)
30	3.44(−5)	3.29(−5)	9.48(−4)	3.0(−5)
	Tilted grid errors			
10	5.83(−4)	2.17(−4)	1.28(−2)	3.6(−5)
16	1.72(−4)	9.20(−5)	5.09(−3)	−1.3(−6)
24	5.06(−5)	2.74(−5)	2.00(−3)	−4.2(−7)
30	2.96(−5)	1.71(−5)	1.19(−3)	−1.6(−7)

TABLE 2

	Main grid errors			Tilted grid errors		
M_1/M_2	u	v	p	u	v	p
10/16	2.26	2.13	2.54	2.60	1.83	1.96
16/24	2.80	2.71	2.66	3.02	2.99	2.30
24/30	2.52	2.10	2.59	2.40	2.11	2.33

in the cartesian coordinates of the larger domain. The origin is taken as the midpoint of the lower boundary of the larger domain. In Table 1 the L^2 norms of the errors for several cases are shown. For these cases, the iteration parameters ω and γ were chosen as $\omega = 2/(1+4\Delta x)$ and $\gamma = 4\Delta x$, where Δx is the x grid spacing on the larger grid. Table 1 also shows the values of d_i, the average divergence, for each domain; see (2.4). The values of d_i are on the order of the error for the computed solution, and thus are acceptable.

The values of ω and γ that were used were found to give about the best convergence rates. Moreover, their dependence on the grid spacing resulted in the number of iterations being roughly proportional to M. The stopping criterion for the iterative methods was to stop when the norms of the changes in the velocities and the norm of the deviation between the divergence of the velocity field and d_i were all sufficiently small. The number of iterations for the cases shown in Table 1 were 179, 327, 527, and 677, respectively, when the L^2 norms of the updates were less than 10^{-5}.

Table 2 gives the convergence rates of the accuracy of successive runs. The convergence rate for runs with values of M equal to M_1 and M_2, with M_1 less than M_2, is

$$\log(\text{error}(M_1)/\text{error}(M_2)) \,/\, \log(M_2/M_1)\,.$$

These experimentally determined convergence rates are consistent with the overall method being second-order accurate.

The second set of numerical experiments were run using the domain shown in Fig. 4. The grid was generated by constructing a cartesian grid on the domain shown in the lower left portion of Fig. 4, and a polar grid on the domain shown in the lower

TABLE 3

	Cartesian errors			
M	u	v	p	d
10	5.31(−3)	4.73(−3)	2.28(−1)	−3.1(−4)
20	1.21(−3)	9.47(−4)	2.19(−2)	−2.5(−5)
30	5.47(−4)	4.31(−4)	7.81(−3)	−8.9(−6)
40	3.12(−4)	2.45(−4)	4.60(−3)	−2.4(−6)

	Polar errors			
M	u	v	p	d
10	2.50(−3)	1.12(−2)	2.17(−1)	2.8(−4)
20	8.19(−4)	2.79(−3)	4.83(−2)	3.3(−5)
30	3.99(−4)	1.22(−3)	2.11(−2)	1.3(−5)
40	2.25(−4)	6.99(−4)	1.22(−2)	7.0(−6)

right portion of Fig. 4. In the top domain shown in Fig. 4, the dotted lines show the overlap between the two domains. In almost all the cases tested Δx and Δy were equal and i_0 and j_0 were also equal.

Several exact solutions were used to test the numerical method. A very useful solution was, in the cartesian representation,

$$u = x, \qquad v = -y, \qquad p = 0, \tag{7.1}$$

or, in polar representation,

$$u = r\cos 2\theta, \qquad v = -r\sin 2\theta, \qquad p = 0. \tag{7.2}$$

This solution has the property that it exactly satisfies the difference approximations to the Stokes equations on both grids of the second problem. Thus the only errors introduced by the method were due to the interpolation. With the use of the finite Fourier expansion for the interpolation from the polar to the cartesian domain, this simple exact solution had essentially no error. The only errors measured were on the order of the arithmetic accuracy of the computer.

The solution for which most of the tests of the second problem were made is

$$\begin{aligned} u &= \tfrac{1}{2}\left(x^2 - y^2\right) / \left(x^2 + y^2\right) - \tfrac{1}{2}\log\left(x^2 + y^2\right), \\ v &= xy/\left(x^2 + y^2\right), \\ p &= 2x/\left(x^2 + y^2\right) + p_0, \end{aligned}$$

in cartesian representation. In Table 3 the L^2 norms of the errors are shown for a representative series of runs. For these runs M and N, as defined by (6.7) and (6.10), were equal. The values of $r_{\min}$ and $r_{\max}$ were 0.2 and 0.9, respectively. The value J was 14 for all runs. For these cases, the iteration parameters ω and γ were chosen as $\omega = 2/(1 + 9\Delta x)$ and $\gamma = 9\Delta x$, where Δx is the x grid spacing on the cartesian grid. As for the first case, these values for ω and γ gave good rates of convergence for the iterative method.

TABLE 4

	Cartesian errors			Polar errors		
M_1/M_2	u	v	p	u	v	p
10/20	2.13	2.32	3.38	1.61	2.01	2.17
20/30	1.96	1.94	2.54	1.77	2.04	2.05
20/40	1.96	1.94	2.25	1.86	1.97	1.99
30/40	1.95	1.96	1.84	1.99	1.94	1.90

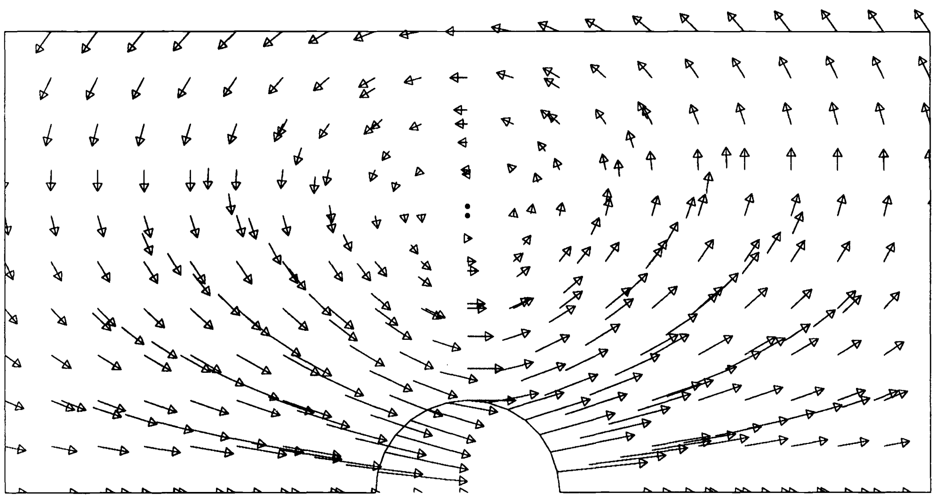


FIG. 5

Table 4 displays the convergence rates for the accuracy of successive runs. The convergence rates were computed as for Table 2. As with the first case, these experimentally determined convergence rates are consistent with the overall method being second-order accurate.

The pressure error norms are computed in a way similar to the computation of a statistical variance, that is, the pressure error norm measures the deviation of the pressure from its mean value. The algorithm that was used is due to West [19]. In this way, the effect of an additive constant is canceled. The norm of the difference between the divergence and d_i was also computed using this algorithm. As used here, d_i corresponds to the statistical average, and the norm of the difference is obtained as the standard deviation.

The velocity field for the run given in Table 3 with M equal to 20 is shown in Fig. 5, and in Fig. 6 the pressure contours are shown. As shown by Fig. 5, the solution provides a good test problem by being nontrivial. Figure 6 was obtained by overlaying the contour plots from each of the domains.

As mentioned in §2 the size of the overlap between the domains affected the error of the solution. For example, if the value of $r_{\max}$ was decreased from 0.9 to 0.6 the errors increased by about 50 percent. The efficiency was also affected by decreasing

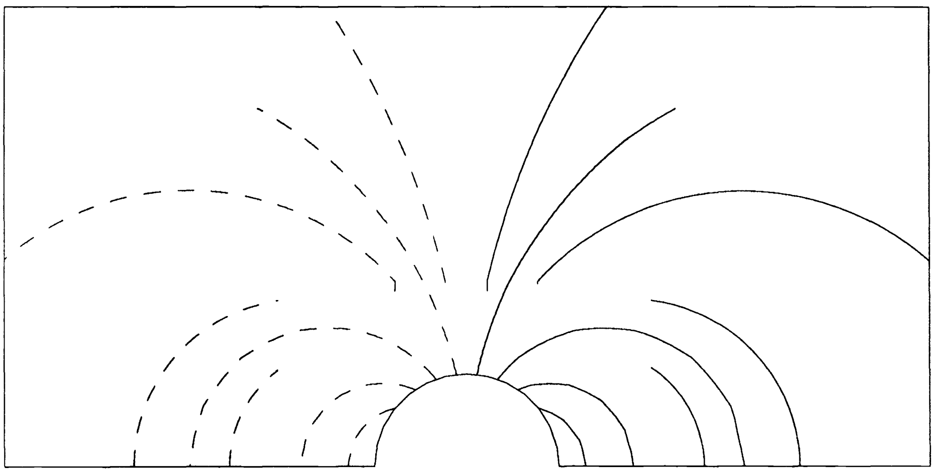


FIG. 6

$r_{\max}$. The number of iterations required to achieve a given convergence criterion was more than four times as great for $r_{\max}$ of 0.6 as for $r_{\max}$ of 0.9. This increase in error when the overlap is decreased is most likely due to the interaction of the errors induced by the interpolation on the boundaries.

8. Conclusions. The domain decomposition method presented here has been demonstrated to compute accurate solutions to the equations describing incompressible viscous flow. The method described here can be extended to computing flows of engineering significance involving the Navier–Stokes equations. It is anticipated that there will be few difficulties encountered in extending this method from the Stokes equations to the Navier–Stokes equations for low Reynolds number flows.

Research is being done to extend this work to cases with more than two subdomains. There are also plans to study the algorithm on parallel architectures to see what speedup can be achieved.

We appreciate the suggestions and corrections made by the referees.

REFERENCES

[1] J. CAHOUET, *On some difficulties occurring in the simulation of incompressible fluid flows by domain decomposition methods*, in Proc. First Internat. Symp. 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, Phila., PA, 1988, pp. 313–332.

[2] G. S. CHESSHIRE, *Composite grid construction and applications*, Ph.D. thesis, Dept. of Applied Math., California Institute of Technology, Pasadena, CA, 1986.

[3] Q. V. DINH, R. GLOWINSKI, J. PÉRIAUX, AND G. TERRASSON, *On the coupling of viscous and inviscid models for incompressible fluid flows via domain decomposition methods*, in Proc. First Internat. Symp. 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, Phila., PA, 1988, pp. 350–369.

[4] M. Fortin and R. Aboulaich, *Schwarz's decomposition method for incompressible flow problems*, in Proc. First Internat. Symp. 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, Phila., PA, 1988, pp. 333–349.

[5] R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Proc. First Internat. Symp. on Domain Decomposition Methods for Partial Differential Equations, Society for Industrial and Applied Mathematics, Phila., PA, 1988.

[6] W. D. Henshaw and G. Chesshire, *Multigrid on composite meshes*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 914–923.

[7] B. Kreiss, *Construction of curvilinear grids*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 270–279.

[8] P. L. Lions, *On the Schwarz alternating method* I, in Proc. First Internat. Symp. 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, Phila., PA, 1988, pp. 1–42.

[9] H. J. Lugt and E. W. Schwiderski, *Flow around dihedral angles* I, *Eigenfunction analysis*, Proc. Roy. Soc. Lond. A, 285 (1965), pp. 382–399.

[10] J. C. Strikwerda, *Finite difference methods for the Stokes and Navier–Stokes equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 56–68.

[11] ———, *An iterative method for solving finite difference approximations to the Stokes equations*, SIAM J. Numer. Anal., 21 (1984), pp. 447–458.

[12] J. C. Strikwerda and D. Shin, *Fast solvers for finite difference approximations for the Stokes and Navier–Stokes equations*, SIAM J. Sci. Comput., submitted.

[13] J. C. Strikwerda and Y. M. Nagel, *A numerical method for the incompressible Navier–Stokes equations in three-dimensional cylindrical geometry*, J. Comp. Phys., 78 (1988), pp. 64–78.

[14] J. C. Strikwerda, B. A. Wade, K. P. Bube, *Regularity estimates up to the boundary for elliptic systems of difference equations*, SIAM J. Numer. Anal., 27 (1990), pp. 292–322.

[15] R. Temam, *Navier–Stokes Equations, Theory and Numerical Analysis*, North-Holland, Amsterdam, 1984.

[16] J. F. Thompson, ed., *Numerical Grid Generation*, Elsevier/North Holland, New York, 1982.

[17] M. C. Thompson and J. H. Ferziger, *An adaptive multigrid technique for the incompressible Navier–Stokes equations*, J. Comp. Phys., 82 (1989), pp. 94–121.

[18] S. P. Vanka, *Block-implicit multigrid solution of Navier–Stokes equations in primitive variables*, J. Comp. Phys., 65 (1986), pp. 138–158.

[19] D. H. D. West, *Updating mean and variance estimates: An improved method*, Comm. ACM, 22 (1979), pp. 532–535.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 68–80, January 1993

SOME PRACTICAL ASPECTS OF EXPLORATORY PROJECTION PURSUIT*

JIAYANG SUN†

Abstract. This paper shows how to choose the number of terms in a projection pursuit index and how to select a projection pursuit algorithm. It also demonstrates that Friedman's index [*J. Amer. Statist. Assoc.*, 82 (1987), pp. 249–266] works better than Hall's [*Ann. Statist.* 17 (1989), pp. 589–605] in detecting separations or clusters.

Key words. projection pursuit, optimizations, P-values, consistency

AMS(MOS) subject classifications. 62H99, 62G99, 62E30, 65C05, 65K10

1. Introduction. Projection pursuit explores interesting "nonlinear structures"[1] (such as clusters and separations) of a high-dimensional data set by projecting the data onto some low-dimensional space. In this paper we denote the high p-dimensional independent and identically distributed (i.i.d.) data set from a population $\mathcal{F}$ by $Y_1, \dots, Y_N$ and pursue the case of projecting to one-dimensional space. (Extension to the case of two dimensions is possible.)

The two key elements in projection pursuit are the related index and algorithm. A *projection pursuit index* $I(\alpha)$ is an estimate of a measure of distance between the distribution of the projection of the data in the direction $\alpha = (\alpha_1, \dots, \alpha_p)$ and some "*uninteresting distribution*." Of course, we only need to consider vector α with $\alpha^T\alpha = 1$. A *projection pursuit algorithm* is an optimization algorithm that maximizes $I(\alpha)$. Since classical statistics can be used to handle the data from a normal distribution well (cf. [1]), a natural uninteresting distribution is a normal distribution. The two recent projection pursuit indices proposed by Friedman [4] and Hall [6] are to find the "least normal" structures.

Let $\Phi(x)$, $\phi(x)$ be the distribution and density function of an $\mathcal{N}(0,1)$ random variable (r.v.), respectively. Denote the density function of $X = \alpha^T Y$ by $p_\alpha(x)$. Assume for the moment that $E(Y_i) = 0$ and $\operatorname{cov}(Y_i) = I$, the identity matrix. The m-term *Hall's index* I_m^H is an estimate of the sum of the first m terms of Hermite polynomial expansion of $\mathcal{L}_2$ distance between p_α and ϕ:

$$(1)\qquad \begin{aligned} I_m^H(\alpha) = &\sum_{j=1}^{m} \frac{\sqrt{\pi}}{j! \cdot 2^{j-1}} \left\{ \frac{1}{N} \sum_{i=1}^{N} H_j(\alpha^T Y_i)\phi(\alpha^T Y_i) \right\}^2 \\ &+ 2\pi^{1/2} \left\{ \frac{1}{N} \sum_{i=1}^{N} \phi(\alpha^T Y_i) - \frac{1}{\sqrt{2}\pi^{1/4}} \right\}^2, \end{aligned}$$

where the Hermite polynomials H_j are defined as follows:

$$H_0(r) = 1, \qquad H_1(r) = 2r, \qquad H_j'(r) = 2jH_{j-1}(r),$$

*Received by the editors September 10, 1990; accepted for publication December 8, 1991.

†Department of Statistics, University of Michigan, 1444 Mason Hall, Ann Arbor, Michigan 48109-1027.

[1]The term "nonlinear structures" refers to those structures that cannot be detected easily by using solely sample mean and covariance matrix of the data, in contrast to classical multivariate statistics.

$$\int_{-\infty}^{\infty} H_j^2(r)\,\phi^2(r)dr = \frac{j!\,2^{j-1}}{\sqrt{\pi}} \quad \text{for } j = 2, 3, \ldots.$$

Under the transformation $X \longrightarrow R = 2\Phi(X) - 1$, the density $p_\alpha(x)$ is transformed to the density function $q_\alpha(r)$ of $R = 2\Phi(\alpha^\tau Y) - 1$, and a standard normal r.v. will be transformed to a uniform r.v. on $(-1, 1)$. The m-term *Friedman's index* I_m^F is an estimate of the sum of the first m terms of Legendre polynomial expansion of $\mathcal{L}_2$ distance between q_α and $\frac{1}{2}$:

$$I_m^F(\alpha) = \sum_{j=1}^{m} \frac{2j+1}{2} \left[\frac{1}{N}\sum_{i=1}^{N} L_j\left\{2\Phi(\alpha^\tau Y_i) - 1\right\}\right]^2, \tag{2}$$

where the recursive definition of Legendre polynomials L_j is given as follows:

$$L_0(r) = 1, \quad L_1(r) = r, \quad L_2(r) = \frac{1}{2}(3r^2 - 1),$$

$$L_j(r) = \frac{1}{j}\left\{(2j-1)rL_{j-1}(r) - (j-1)L_{j-2}(r)\right\} \quad \text{for } j = 3, \ldots.$$

In practice, $E(Y_i)$ and $\operatorname{cov}(Y_i)$ are unknown and arbitrary. People usually *sphere* the data before implementing the projection pursuit procedure based on the first set of summary statistics: sample mean $\overline{Y} = 1/N \sum Y_i$ and sample covariance matrix $\hat{\Sigma} = (1/N)\sum(Y_i - \overline{Y})(Y_i - \overline{Y})^\tau$. A typical version of the sphered data is

$$Z_i = \hat{D}^{-\frac{1}{2}}\hat{U}^\tau(Y_i - \overline{Y}), \qquad i = 1, \ldots, N, \tag{3}$$

where $\hat{U}\,\hat{D}\,\hat{U}^\tau = \hat{\Sigma}$ is an eigenvalue-eigenvector decomposition of $\hat{\Sigma}$. In other words, Y_i in (1) or (2) is often replaced by its sphered version, say Z_i, in practice.

In §2, the rules of thumb for choosing the appropriate number m of terms in the Friedman and Hall indices are given. It turns out that Friedman's index performs better than Hall's in detecting separations or clusters. In §3, projection pursuit algorithms are discussed for three typical situations: highly structured data, data from Gaussian noise, and none of the above.

2. How to choose the number of terms. The choice of the number m of terms included in a projection pursuit index depends on the data dimension p and sample size N. We give some general guidelines here.

Case 1. *Friedman's index.* Friedman [4] suggested that we choose

(a) $2 \le m \le 6$ in most cases.

In the following we show that m should be at least 3.

Let

$$\tilde{Y}_j(\alpha) = \sqrt{\frac{2j+1}{N}} \sum_{i=1}^{N} L_j(2\Phi(\alpha^\tau Z_i) - 1), \tag{4}$$

for $j = 1, \ldots, m$. Friedman's index can be rewritten as $I_m^F(\alpha) = (2N)^{-1}\sum(\tilde{Y}_j(\alpha))^2$. According to the large sample theory, in the unsphered data case $(Z_i = Y_i)$, these

$\tilde{Y}_j(\alpha)$, $j = 1, 2, \ldots, J$, are i.i.d. univariate standard normal random variables. Note that the sphered version $Z_1, \ldots, Z_N$ of data $Y_1, \ldots, Y_N$ satisfies

$$\sum_{i=1}^{N} Z_i = 0, \qquad \frac{1}{N}\sum_{i=1}^{N} Z_i Z_i^T = I_p, \tag{5}$$

and concentrates on a small neighborhood of 0 with a high probability, in which function $\Phi(x)$ can be approximated by a straight line $ax + \frac{1}{2}$. Thus $R_i = 2\Phi(\alpha^T Z_i) - 1 \approx 2a\alpha^T Z_i$. Hence the first term of Friedman's index $I_m^F(\alpha)$ is

$$\tilde{Y}_1(\alpha)^2 \approx \frac{3}{2}\left(\frac{1}{N}\sum_{i=1}^{N} 2a\alpha^T Z_i\right)^2 = 0.$$

The variance of the second term is also very small relative to the variances of other terms due to the second constraint in (5). We have done simulations to confirm this phenomenon. For example, in Table 1 the data are four-dimensional pseudo-normal random vectors from $\mathcal{N}(0, I_4)$, the sample size is N, and the simulation size M is 1000. We first compute $\tilde{Y}_j(\alpha)$ based on the sphered data for $j = 1, 2, 3, 4$ for each simulation and then calculate the sample mean and variance of $\tilde{Y}_j(\alpha)$. The sample mean and variance of the 1000 pseudounivariate standard normal random numbers (labeled st. normal) are also provided in Table 1.

TABLE 1
$\tilde{Y}_l(\alpha)$ in sphered case.

	$\alpha = (0, 3^{-1/2}, 3^{-1/2}, 3^{-1/2})$				$\alpha = (1,0,0,0)$	
	N=100		N=1000		N=1000	
	var	mean	var	mean	var	mean
$\tilde{Y}_1(\alpha)$	0.0414	0.0147	0.04	0.003	0.045	0.003
$\tilde{Y}_2(\alpha)$	0.22	-0.004	0.22	0.06	0.23	0.08
$\tilde{Y}_3(\alpha)$	0.92	-0.080	0.91	-0.03	0.98	-0.033
$\tilde{Y}_4(\alpha)$	0.853	0.0490	0.811	-0.04	0.89	-0.0856
st. normal	0.98503	0.061	0.98503	0.061	0.98503	0.061

In Table 1, the means of $\tilde{Y}_l(\alpha)$'s are reasonably close to zero; the variances of $\tilde{Y}_3(\alpha)$ and $\tilde{Y}_4(\alpha)$ are close to one, but the variance of $\tilde{Y}_1(\alpha)$ is only about 0.04, and the variance of $\tilde{Y}_2(\alpha)$ is about 0.22, which is closer to 0 rather than 1. Intuitively speaking, sphering reduces dimensionality of the projection pursuit index by 2. A theoretical justification can be found in the appendix. Therefore, we suggest modifying (a) as

(a′) $3 \le m \le 6$ in most cases.

Recall that Friedman's index, for example, estimates the sum of the first m terms of Legendre polynomial expansion of $\mathcal{L}_2$ distance between q_α and $\frac{1}{2}$ by substituting $E(L_j\{2\Phi(\alpha^T Y) - 1\})$ with the corresponding sample mean $1/N \sum L_j\{2\Phi(\alpha^T Y_i) - 1\}$ (cf. Friedman [4]). The $L_j\{2\Phi(\alpha^T Y) - 1\}$ all have similar variance; indeed, under normality, all have variance close to 1 for $j \ge 3$. However, if the $\mathcal{L}_2$ distance between distributions is finite, the means must converge to 0 as j increases. This implies that the L_j for large j will be poor at detecting nonnormality, especially when N is small. In practice, one may always assume a large finite support. Then the corresponding L_2 distance is finite. This assumption does not change the interesting structures of the data. Thus,

(b) if N is small, m should also be small.

The N data points in a higher-dimensional space suffer more from "the curse of the dimensionality" than the N data points do in a lower-dimensional space. In other words, this case can be treated as if the sample size is small. Consequently,

(c) the larger p is, the smaller m should be.

In summary, (a′), (b), and (c) form a rule of thumb for choosing the number of terms for Friedman's projection pursuit index. Less experienced users may find the examples in Friedman [4] useful.

Case 2. *Hall's index*. We discuss a motivating example first.

The bimodal model is an interesting model in applications. Suppose that the data $Y_1, \ldots, Y_N$ are i.i.d. from a four-dimensional population $\mathcal{G}_\mu$, which specifies that if $Y_i = (Y_1^i, Y_2^i, Y_3^i, Y_4^i)^\tau \overset{\text{dis}}{\sim} \mathcal{G}_\mu$, then

- the first component Y_1^i is a r.v. with the density function g_μ:

$$g_\mu(x) = \tfrac{1}{2}\phi(x) + \tfrac{1}{2}\phi(x-\mu), \qquad x \in (-\infty, \infty); \tag{6}$$

- the remaining components Y_2^i, Y_3^i, Y_4^i are i.i.d. from $\mathcal{N}(0,1)$;

and $Y_1^i, Y_2^i, Y_3^i, Y_4^i$ are independent. The truly interesting direction for this probability model is $\alpha^* = (1,0,0,0)$.

We shall compare the four-term Friedman index ($m = 4$) with the four-term Hall index under the above simple probability model in the case of sample size $N = 100$. In Hall's index, there are five terms if we count the zero term. The comparison will be given in terms of histograms explained below:

$$\begin{tikzcd} & & \arrow[r, squiggly] & \alpha_m^F \arrow[r, "\text{project}"] & \alpha_m^{F\tau} Z \arrow[r] & \text{histogram}_F \\ Y \arrow[r, "\text{sphere}"] & Z \arrow[ur] \arrow[dr] & & & & \\ & & \arrow[r, squiggly] & \alpha_m^H \arrow[r, "\text{project}"] & \alpha_m^{H\tau} Z \arrow[r] & \text{histogram}_H \end{tikzcd} \tag{7}$$

Here in (7) for each data Y from a bimodal model, we sphere Y to get the sphered version Z of Y, then obtain two different histograms in two different ways. Based on the data Z, one way is to run a projection pursuit algorithm with Friedman's index as the objective function to obtain a solution α_m^F, then take the histogram of the univariate data $\alpha_m^{F\tau} Z$ as an estimate of the density function of $\alpha_m^{F\tau} Z$. Another way is to run the same algorithm based on the same data Z with Hall's index as the objective function to reach a solution α_m^H, then graph the histogram of the data $\alpha_m^{H\tau} Z$ to estimate its density function. We interpret the projection pursuit index as performing well if the corresponding histograms are close to the density function of the original bimodal model, and/or its solution (α_m^F or α_m^H) is close to $\alpha^* = (1,0,0,0)$.

When μ in (6) is small, we may not be able to see a separation of two peaks from the raw data even if they are projected onto α^*. Therefore, we construct $Y_i = (Y_1^i, \ldots, Y_4^i)$'s independently as follows. Let W_1, W_2, W_3, W_4 be four i.i.d. $\mathcal{N}(0,1)$ pseudo r.v.'s which are independent of a uniform r.v. U on $(0,1)$. Define

$$Y_1^i = W_1 + \mu I_{\{U > \frac{1}{2}\}}, \quad Y_2^i = W_2, \quad Y_3^i = W_3, \quad Y_4^i = W_4.$$

For each set of $W_1, \ldots, W_4$, we consider an increasing sequence of μ's. Here $\mu = 0.4, 0.8, \ldots, 7.6, 8$.

Figure 1 presents the results of our first experiment where the histograms are of the sphered data projected on α_m^F and α_m^H in the sphered space for $\mu = 0.4, 0.8, \ldots, 8.0$. The only difference between these histograms and the histograms of the unsphered

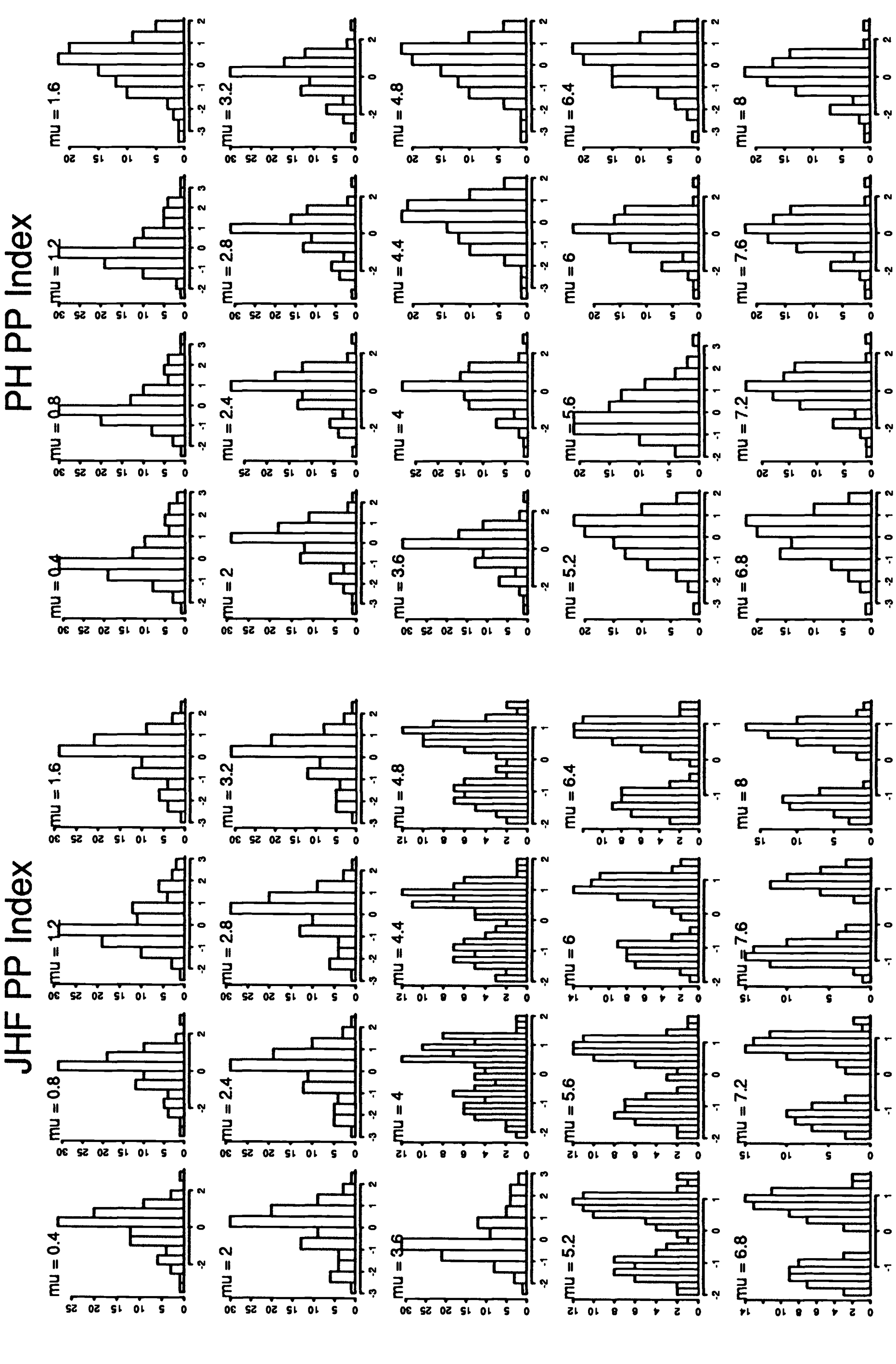


FIG. 1. *Histograms of the data projected on* α_m^F *and* α_m^H, I.

data projected to the projection pursuit solution in the original space is the scale. The separation is preserved in the sphered space. We see that with Friedman's index, the separation starts from $\mu = 4.4$, while with Hall's index no separation appears in any of these cases with μ chosen from $0.4, 0.8, \ldots, 8.0$. In fact, if we check the directions $\alpha_m^F(\mu)$ found by using Friedman's index, we notice that $\alpha_m^F(\mu)$'s *are* close to the line represented by the direction $\alpha^* = (1,0,0,0)$ for $\mu = 4.0, 4.4, \ldots, 8.0$, while for $\mu = 0.4, 0.8, \ldots, 8.0$, $\alpha_m^H(\mu)$ are *not*. Note that $(-1,0,0,0)$ and $(1,0,0,0)$ give the same straight line.

Figure 2 shows the second experiment, which is independent of the first one. We see separations starting from $\mu = 3.6$ up to 8.0 by applying Friedman's index, and separations at $\mu = 4.8$ and 6.0 up to 8.0 by using Hall's index.

Table 2 reveals that the lines represented by directions found by using Friedman's index are close to 1 when $\alpha^* = (1,0,0,0)$ for $\mu = 3.2$ up to 8.0. Those found by using Hall's index are close to $\alpha^* = (1,0,0,0)$ for $\mu = 4.8$ and 6.0 up to 8.0.

TABLE 2
The directions found by the Friedman and Hall indices.

μ	$\alpha_m^F(\mu)$				$\alpha_m^F(\mu)$			
0.4	(0.71285,	0.36443,	-0.45505,	-0.38984)	(0.084005,	-0.20921,	-0.56222,	-0.79567)
0.8	(-0.25511,	-0.006226,	0.69983,	-0.66717)	(0.19501,	-0.24838,	-0.64503,	0.69585)
1.2	(-0.47785,	-0.037650,	-0.61193,	0.62912)	(0.43953,	0.091376,	0.66845,	-0.59299)
1.6	(-0.51711,	0.0045139,	0.64914,	0.55785)	(0.52236,	0.096520,	-0.67104,	-0.51724)
2.0	(-0.59760,	0.035192,	0.60462,	0.52543)	(0.53695,	0.17745,	-0.67581,	-0.47273)
2.4	(0.23240,	-0.74695,	0.27143,	-0.56070)	(0.12243,	-0.21050,	-0.14695,	-0.95870)
2.8	(-0.30959,	-0.74266,	-0.21769,	-0.55247)	(0.18994,	0.22677,	-0.14095,	0.94479)
3.2	(0.98420,	-0.016513,	0.053772,	-0.16786)	(0.24278,	-0.24563,	-0.13253,	-0.92907)
3.6	(0.98719,	0.0074661,	0.062255,	-0.14674)	(-0.85279,	-0.37562,	0.35878,	0.054219)
4.0	(-0.99076,	0.034163,	-0.042933,	0.12402)	(0.41525,	0.40581,	0.67988,	-0.44794)
4.4	(0.99263,	-0.034992,	0.042527,	-0.10798)	(0.34723,	-0.30182,	-0.10023,	-0.88221)
4.8	(-0.99407,	0.034186,	-0.041981,	0.094286)	(1.0000,	0.,	0.,	0.)
5.2	(0.99519,	-0.032652,	0.041025,	-0.082791)	(0.39119,	-0.33279,	-0.079074,	-0.85438)
5.6	(0.99604,	-0.030866,	0.039724,	-0.073298)	(0.40862,	-0.34610,	-0.069336,	-0.84168)
6.0	(0.99670,	-0.029068,	0.038212,	-0.065528)	(1.0000,	0.,	0.,	0.)
6.4	(0.99720,	-0.027368,	0.036605,	-0.059177)	(1.0000,	0.,	0.,	0.)
6.8	(0.99760,	-0.025804,	0.034987,	-0.053960)	(1.0000,	0.,	0.,	0.)
7.2	(0.99791,	-0.024385,	0.033408,	-0.049639)	(1.0000,	0.,	0.,	0.)
7.6	(0.99816,	-0.023104,	0.031899,	-0.046021)	(1.0000,	0.,	0.,	0.)
8.0	(0.99837,	-0.021948,	0.030472,	-0.042960)	(1.0000,	0.,	0.,	0.)

We have printed out 20 experiments, of which the first two are included in this paper. All of these 20 experiments show the same pattern as the first two. Friedman's four-term index is superior to Hall's four-term index for this bimodal model.

Explanations. By Cramér [2, p. 133 and pp. 221–224], the statistics formed from the third Hermite polynomial give the skewness of the data, and the statistics formed from the fourth Hermite polynomial give the kurtosis of the data. Recall the argument in Case 1 that if the data are sphered, with high probability, they are concentrated on a small neighborhood of zero. Therefore, in the four-term Hall's index, the zeroth, first, and second terms do not change much, as they have much smaller variances than those of the third and fourth terms. The third term is dominated by the skewness of

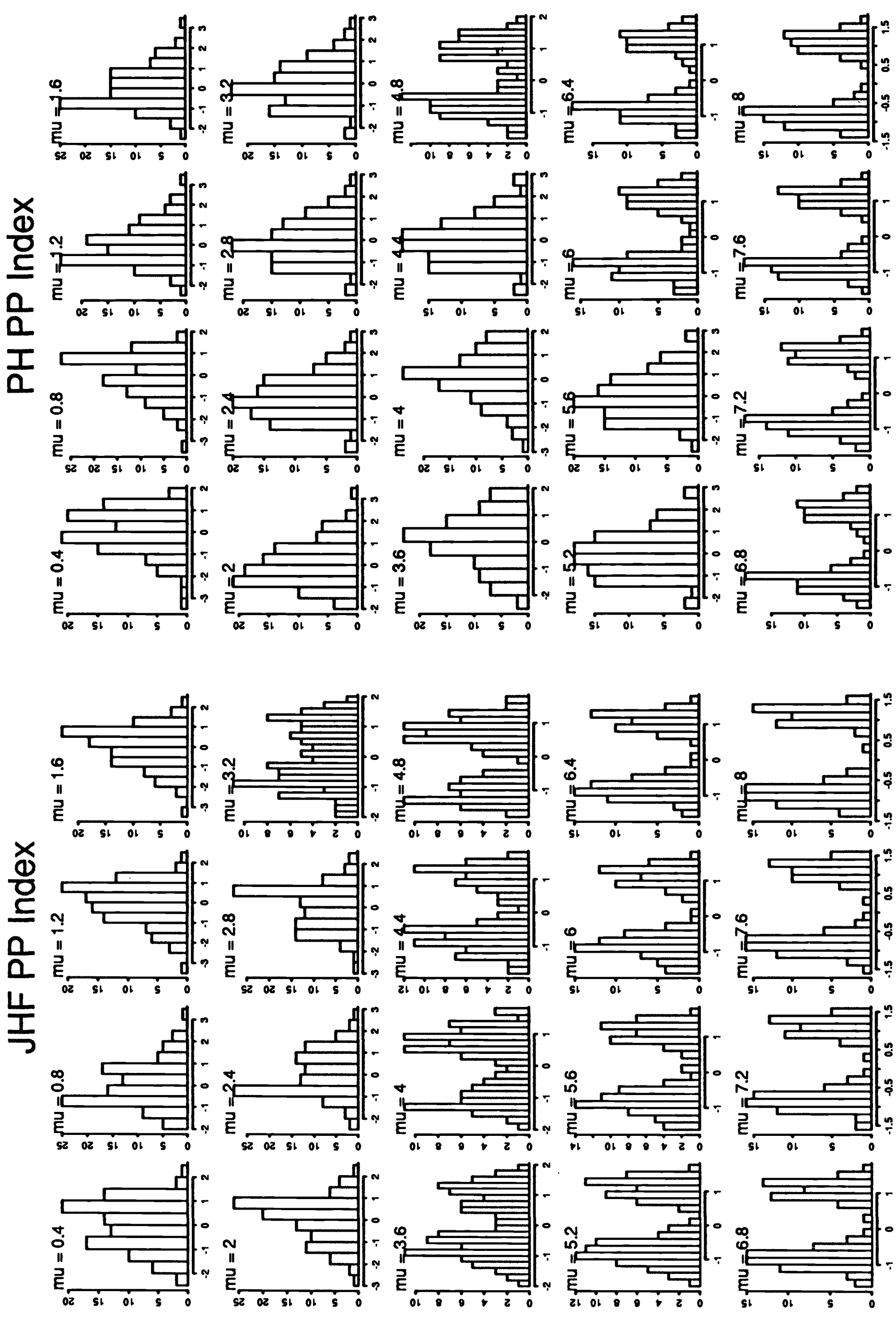


FIG. 2. *Histograms of the data projected on* α_m^F *and* α_m^H, II.

the data, and the fourth term by the kurtosis of the data. In our experiments, the four-term Hall's index places too much weight on the skewness of the data and picks up views with skewed density rather than bimodal density in many cases.

If more terms in Hall's index are used, one might get the same result as Friedman's index does for this model. In fact, we can make a connection between Hall's consistency result [6], and the number of terms that should be chosen in a projection pursuit index.

Denote by α^H the true maxima of the $\mathcal{L}_2$ distance between the densities p_α and ϕ and by α^F the true maxima of the $\mathcal{L}_2$ distance between the transformed densities $q_\alpha(\cdot)$ and $\frac{1}{2}$. Let α_m^H and α_m^F be the projection pursuit solutions of the indices $I_m^H(\alpha)$ and $I_m^F(\alpha)$. Under some regularity conditions for the density function of the data, the consistency result says that as $N, m \to \infty$,

$$\begin{aligned} \alpha_m^F - \alpha^F &= O_p(N^{-1/2}), \quad \text{if } m/N^{1/3} \to 0, \quad m/N^{1/4r} \to \infty, \\ \alpha_m^H - \alpha^H &= O_p(N^{-1/2}), \quad \text{if } m/N^{2/3} \to 0, \quad m/N^{1/2r} \to \infty, \end{aligned}$$

where $r \geq 1$. This result shows that the number of terms in Hall's index should be about the square of the appropriate number of terms in Friedman's index. In our experiments, sample size $N = 100$, $N^{\frac{1}{3}} = 4.64$, $N^{\frac{1}{4}} = 3.16$. According to the above consistency result, $m = 4$ in Friedman's index is reasonable, while $m = 16$ in Hall's index might give as good a result as the four-term Friedman index. In summary, a rule of thumb for choosing the number of terms in Hall's index is

(a) $3 \leq m \leq 36$ in most cases;
(b) the smaller N is, the smaller m should be;
(c) the bigger p is, the smaller m should be.

Computationally, the maximization of the higher-order polynomial objective function is much more expensive than the maximization of the lower-order polynomial objective function. If we include more terms in Hall's index to achieve the same accuracy of Friedman's index with fewer terms, some computational efficiency is lost.

Remark 1. It is easy to see that this loss of dimensionality due to sphering occurs for all polynomially based projection pursuit indices for any versions of sphering that satisfy (5). Hence the minimum number of terms for all such indices should be three.

3. How to select an algorithm. The primary goal of a projection pursuit algorithm is to find the direction which maximizes a projection pursuit index. Just like any other optimization algorithm, a projection pursuit algorithm is usually well suited to a particular type of problem, or a particular combination of projection pursuit index and data structure. By a naive (use of) algorithm, the computed value for the observed $\max I_J(\alpha)$ can be merely a local maximum of $I_J(\alpha)$, which is far away from the global maximum in some important cases. On the other hand, a projection pursuit procedure should find multiple views of a multivariate data set, i.e., it should enable one to keep running the algorithm based on some index until as many informative directions as possible are discovered (cf. Friedman [4]). Therefore, a good projection pursuit algorithm is one that can be used to find the first several largest local maxima in a reasonable time.

Friedman [4] gave a practical projection pursuit algorithm for his index. This algorithm has many computational advantages and uses an interesting idea called *structure removal*. Since the normal structure is the "least interesting" and is associated with directions that minimize the index, the idea of the structure removal is to transform the data along the "interesting" direction (found by the algorithm) into standard normal data while keeping the structure along the other orthogonal directions unchanged. The structure removal ensures that the interesting directions already found will not be found again later.

Friedman's projection pursuit algorithm can be described briefly as follows.

1. *Preliminary search*: search along p coordinates (or principle components in the sphered case); the one with largest projection pursuit index is called $\alpha^{(1)}$.
2. *Coarse search*: large stepping search along p orthogonal directions with origin at $\alpha^{(1)}$; the one with largest projection pursuit index is called $\alpha^{(2)}$.
3. *Local search*: starting from $\alpha^{(2)}$, search to a convergence point by using steepest ascent or quasi-Newton algorithm as the local optimizer. The direction with largest projection pursuit index after convergence of the algorithm is called $\alpha^{(3)}$.
4. *Structure removal*: transform the data along $\alpha^{(3)}$ into a standard normal data.
5. *Loop*: repeat above steps 2–4.

This algorithm is available from STATLIB.[2]

In the following, we shall discuss Friedman's algorithm and its modifications in three different situations: (a) the data set is highly structured; (b) the data set is pure noise; (c) neither (a) nor (b).

(a) *Data set is highly structured*. In this case, the objective function, i.e., the projection pursuit index based on the data, has a few big peaks surrounded by many small peaks. There is no doubt that Friedman's index performs well in this case.

Two questions arise naturally. First, How many structure removals should be performed? A small P-value associated with the current direction, or projection pursuit solution, signals the stop of structure removals. Then the problem becomes how to calculate the P-values, which is discussed in the next paragraph. Second, What happens if the data set is not highly structured? This is discussed in (c), later in this section.

(b) *Data set is pure noise*. This case occurs if (i) the data is just a result of Gaussian noise; (ii) the P-value of an observed projection pursuit solution is computed by using the Monte Carlo method. Specifically, the P-value is

$$P_o = P\left\{\max_{\alpha\in S^{p-1}} I_m(\alpha) \geq I^*_{\text{obs}}\right\} \tag{8}$$

under H_0' : Y_i 's are i.i.d. $\mathcal{N}(\mu, \Sigma)$ random samples with some p-dimensional vector μ, $p\times p$ nonsingular matrix Σ. Here $I_m(\alpha)$ is based on the sphered version Z_i's of Y_i's and $I^*_{\text{obs}} = I_m(\alpha^*) = \max_{\alpha\in S^{p-1}} I_m(\alpha)$ is the index evaluated at the observed projection pursuit solution. This P-value calculation gives a calibration for how large "large" is, or how much structure can be seen from a particular projection pursuit solution.

[2]This is a system for distributing statistical software and data. To obtain the program, contact `statlib@temper.stat.cmu.edu`, then request the algorithm: **send projpur from general**. Do not include a "subject."

When data is unstructured, the objective function $I_m(\alpha)$ has many bumps. Therefore, using Monte Carlo methods to estimate the P-value involves the repeated maximization for functions which have many local maxima with a similar size. Hence the corresponding optimization problem becomes very complicated and computationally intensive. Sun [8] has an easily applicable analytical formula for calculating P-values associated with Friedman's index. We recommend using this formula as a conservative measurement.

Day [3] showed apparent structures that can occur in projections of multivariate normal data. To ensure that apparent structures of the noise data are found, it is helpful to use the more thorough optimization procedure described in (c).

(c) *Neither of the above extreme cases*. In general, the data set may be neither pure noise nor highly structured. Hence, it is useful to improve computational speed, reliability, and accessibility of the projection pursuit algorithm so that the result is reliable for a much wider class of data structures.

Gill, Murray, Michael, and Wright [5] developed several refined versions of an optimization algorithm called NPSOL, which uses a sequential quadratic programming algorithm. It is considered one of the most efficient algorithms among the existing local optimizers. We have done many simulations to confirm its efficiency and reliability in the context of projection pursuit. NPSOL is now available as a subroutine called EO4UCF in NAG (mark 13). We suggest that the local optimizer in the third step of Friedman's algorithm be replaced by NPSOL (version 4.0 or the latest version).

If someone has the old IMSL (release 9.2), the subroutine called ZXMWD can be used as a substitute for steps 1–3 in Friedman's algorithm. With this substitution, the modified Friedman algorithm does slightly better than the original when no structure removal is used in the noise situation, because ZXMWD is a more exhaustive search procedure.

Friedman's idea of structure removal can be used as an enhancement to obtain the global maximum or directions close to it. The direction that maximizes the projection pursuit index among all the directions examined in the steps of the algorithm above can be regarded as the global maximum. Our experiment shows that p times structure removal is sufficient for a p-dimensional data set.

Since an optimization algorithm may not be rotation invariant, adding a *rotation* technique for the Monte Carlo simulation of P-value in projection pursuit is usually beneficial. The idea of the rotation is to run the same algorithm again for the rotated data set Y''s from the original data set Y's.

In summary, we suggest enhancing Friedman's algorithm in practice by the following steps.

3. *Local search*: replace the local optimizer by NPSOL.

5. *Loop* 1: repeat steps 2–4 p **times**.

6. *Loop* 2: **rotate** the original data and repeat steps 2–5. The resulting direction, which has the maximum index among all the above directions, is our first interesting direction.

7. *Next view*: if the associated P-value of the solution derived in step 6 is large, stop. Otherwise, remove the structure along the solution. Then repeat steps 1–6 to obtain the next interesting view.

Sun [7] has done many simulations. The theoretical P-value approximation to (8) (Sun [8]) is very good for the first view obtained from steps 1–6. In other words, we are confident that the algorithm suggested above works for a wide class of functions because it works even for the noise data case. The views collected from steps 6 and 7 are close to the most informative multiple views of a multivariate data set.

Remark 2. The idea of structure removal can be implemented in the maximization of an objective function that has known minima. Hence it has great potential, like simulated annealing in the field of optimization.

Remark 3. The projection pursuit procedure should be used only after some preliminary analysis and "structure removal" are done. For example, the sample mean and covariance matrix can serve as a good first set of summary statistics. Sphering the data removes the structure due to these first and second moments. If one or more of the variables are categorical, they show obvious structure and can be removed from the data set before implementing the projection pursuit procedure. See Friedman [4] for more discussion of these points.

Appendix. A theoretical justification. Notation here is the same as in previous sections. For example, $Y_1, \dots, Y_N$ are the data and $Z_1, \dots, Z_N$ are the sphered version of Y_i's defined in (3). In this appendix, we show that as $N \to \infty$ each $\tilde{Y}_j^s(\alpha)$ is distributed asymptotically according to a univariate normal population $\mathcal{N}(0, \sigma_j^2)$ for $j = 1, 2, \dots, J$, where σ_1^2, σ_2^2 are close to zero and $\sigma_3^2, \dots, \sigma_J^2$ are close to one. This gives a theoretical justification to the dimension reduction argument in Friedman's index. The idea is to apply Taylor expansions of $\tilde{Y}_j^s(\alpha)$ about $\alpha^\tau Z_i$'s at $\alpha^\tau \tilde{Z}_i$'s.

Without loss of generality, let $\Sigma = \sigma^2 I_p$ for a positive scalar σ^2 and a $p \times p$ identity matrix I_p. Denote $Y_i = (Y_{1i}, \dots, Y_{pi})^\tau$, $\bar{Y}_j = N^{-1}(Y_{j1} + \dots + Y_{jN})$, $\mu = (\mu_1, \dots, \mu_p)^\tau$, and $Z_i = (Z_{1i}, \dots, Z_{pi})^\tau$. Then

$$\tilde{Z}_i = (Y_i - \mu)/\sigma, \qquad \tilde{Z}_{1i} = (Y_{1i} - \mu_1)/\sigma. \tag{9}$$

Set $\hat{D}^{-1/2}\hat{U}^\tau = (\breve{\sigma}_{ij})_{p\times p}$. In Anderson [1], Theorem 3.4.4 and the paragraphs following it give

$$(\breve{\sigma}_{ij})_{p\times p} - I_p/\sigma = O_p\,(N^{-1/2}). \tag{10}$$

As usual, here and in the following, we mean "$O_p(\cdot)$ as $N \to \infty$" by $O_p(\cdot)$. Again, without loss of generality, let α be $(1, 0, \dots, 0)^\tau$ in $\tilde{Y}_j^s(\alpha)$'s. Then

$$\alpha^\tau Z_i = Z_{1i} = \sum_{j=1}^{p} \breve{\sigma}_{1j}(Y_{ji} - \bar{Y}_j) = (Y_{1i} - \bar{Y}_1)/\hat{\sigma}_{11} + R_i, \tag{11}$$

where $\hat{\sigma}_{11} = 1/\breve{\sigma}_{11}$, and $R_i = \sum_{j=2}^{p} \breve{\sigma}_{1j}(Y_{ji} - \bar{Y}_j)$. Equation (10) suggests that $\breve{\sigma}_{ij} = O_p(N^{-1/2}), \hat{\sigma}_{1j} = \sigma + O_p(N^{-1/2})$ for $i \neq 1$. Hence, as the distribution of $Y_{ji} - \bar{Y}_j$ is independent of i, it is easy to see that

$$R_i = O_p(N^{-1/2}) \quad \text{uniformly for } i = 1, \dots N, \tag{12}$$

and these Z_{1i}'s are close to $\tilde{Z}_{1i}$:

$$Z_{1i} - \tilde{Z}_{1i} = \frac{(Y_{1i} - \mu_1)(\sigma - \hat{\sigma}_{11})}{\sigma\hat{\sigma}_{11}} - \frac{(\bar{Y}_1 - \mu_1)}{\hat{\sigma}_{11}} + R_i = O_p(N^{-1/2}). \tag{13}$$

We now derive the asymptotic distribution of the first term $\tilde{Y}_1^s(\alpha)$. From (11)–(13), we have the Taylor expansion

$$\tag{14} \begin{aligned} \tilde{Y}_1^s(\alpha) = \tilde{Y}_j^u(\alpha) &+ \left(\frac{3}{N}\right)^{\frac{1}{2}} \sum_{i=1}^{N} \left\{ \frac{(Y_{1i}-\mu_1)(\sigma-\hat{\sigma}_{11})}{\sigma\hat{\sigma}_{11}} 2\phi\left(\frac{Y_{1i}-\mu_1}{\sigma}\right) \right\} \\ &- \left(\frac{3}{N}\right)^{\frac{1}{2}} \sum_{i=1}^{N} \left\{ \frac{(\bar{Y}_1-\mu_1)}{\hat{\sigma}_{11}} 2\phi\left(\frac{Y_{1i}-\mu_1}{\sigma}\right) \right\} + R + O_p\left(N^{-1/2}\right), \end{aligned}$$

where $R = (3/N)^{\frac{1}{2}} \sum [R_i 2\phi\{(Y_{1i}-\mu_1)/\sigma\}]$. From $E[(Y_{ji}-\mu_j)2\phi\{(Y_{1i}-\mu_1)/\sigma\}] = 0$ for all j and (12), we see

$$R = O_p\left(N^{-1/2}\right), \qquad N^{-1} \sum_{i=1}^{N} \left\{ (Y_{1i}-\mu_1) 2\phi\left(\frac{Y_{1i}-\mu_1}{\sigma}\right) \right\} = O_p\left(N^{-1/2}\right).$$

Hence, by $N^{\frac{1}{2}}(\sigma - \hat{\sigma}_{11}) = O_p(1)$, the second term in (14) is $O_p\left(N^{-1/2}\right)$. As

$$(2/N) \sum \phi\{(Y_{1i}-\mu_1)/\sigma\} = 1/\pi^{\frac{1}{2}} + O_p(N^{-1/2}),$$

the third term in (14) is

$$\left(\frac{3}{N}\right)^{\frac{1}{2}} \sum_{i=1}^{N} \left\{ \frac{(\bar{Y}_1-\mu_1)}{\sigma} 2\phi\left(\frac{Y_{1i}-\mu_1}{\sigma}\right) \right\} = R_{1s} + O_p\left(N^{-1/2}\right),$$

where $R_{1s} = (3/N)^{\frac{1}{2}} \sum (Y_{1i}-\mu_1)/(\sigma\pi^{\frac{1}{2}})$. Therefore,

$$\tilde{Y}_1^s(\alpha) = \tilde{Y}_1^u(\alpha) - R_{1s} + O_p\left(N^{-1/2}\right),$$

which can be easily distributed asymptotically as a $\mathcal{N}(0, \sigma_1^2)$ random variable with $\sigma_1^2 = 0.04507$.

Similarly, we can show that as $N \to \infty$ each $\tilde{Y}_j^s(\alpha)$ is distributed asymptotically as a $\mathcal{N}(0, \sigma_j^2)$ random variable for $j = 2, \ldots, J$, where σ_j^2 are as follows:

$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$
0.2421419	0.9665128	0.8642244	0.9933252	0.9536364	0.9977218	0.9788649

Acknowledgments. The author is grateful to David Siegmund, Iain Johnstone, Jerome Friedman, and Clive Loader for helpful discussions and suggestions. Thanks also go to the associate editor and the referees for nice comments, which made the current revision possible.

REFERENCES

[1] T. W. Anderson, *An Introduction to Multivariate Statistics*, 2nd ed., John Wiley, New York, 1984.

[2] H. Cramér, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, NJ, 1954.

[3] N. E. Day, *Estimating components of a mixture of normal distributions*, Biometrika, 56 (1969), pp. 463–474.

[4] J. H. FRIEDMAN, *Exploratory projection pursuit*, J Amer. Statist. Assoc., 82 (1987), pp. 249–266.

[5] P. E. GILL, W. MURRAY, A. S. MICHAEL, AND M. H. WRIGHT, *User's guide for* NPSOL (*Version* 4.0), Tech. Rep., Systems Optimization Laboratory, Dept. of Operations Research, Stanford Univ., Stanford, CA, 1986.

[6] P. HALL, *Polynomial based projection pursuit*, Ann. Statist., 17 (1989), pp. 589–605.

[7] J. SUN, P-*values in projection pursuit*, Tech. Rep., Dept. of Statistics, Stanford Univ., Stanford, CA, August 1989.

[8] ———, *Significan :e levels in exploratory projection pursuit*, Biometrika, 78 (1991), pp. 759–769.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 81–94, January 1993

PARALLEL ALGORITHMS AND SUBCUBE EMBEDDING ON A HYPERCUBE*

ELEANOR CHU†‡ AND ALAN GEORGE†

Abstract. It is well known that the connection in a hypercube multiprocessor is rich enough to allow the embedding of a variety of topologies within it. For a given problem, the best choice of topology is naturally the one that incurs the least amount of communication and allows parallel execution of as many tasks as possible. In a previous paper we proposed efficient parallel algorithms for performing QR factorization on a hypercube multiprocessor, where the hypercube network is configured as a two-dimensional subcube-grid with an aspect ratio optimally chosen for each problem. In view of the very substantial *net* saving in execution time and storage usage obtained in performing QR factorization on an optimally configured subcube-grid, similar strategies are developed in this work to provide highly efficient implementations for three fundamental numerical algorithms: Gaussian elimination with partial pivoting, QR factorization with column pivoting, and multiple least squares updating. Timing results on Intel iPSC/2 and iPSC/860 hypercube multiprocessors are reported for all three algorithms.

Key words. parallel computation, hypercube multiprocessors

AMS(MOS) subject classifications. 65F05, 65F50, 68R10

1. Introduction.

This paper is a sequel to [3], where we propose medium-grain parallel algorithms for hypercubes with substantial startup cost and message-passing latency. This class of hypercube multiprocessors includes those commercially available from Intel and NCUBE. It is well known in practice and evident in the literature that fine-grain algorithms are designed for hypercubes with no startup cost and high communication bandwidth, and that their performance and analytical results do not carry over to medium-grain parallel algorithms.

In [3] we described medium-grain QR factorization algorithms that can dynamically adapt to topological reconfiguration and achieve substantial saving in time and storage. The topology we advocate is very closely related to the two-dimensional mesh and torus, but enjoys a number of distinct features important in designing efficient parallel algorithms. We obtain this topology by configuring the hypercube network as a two-dimensional *subcube-grid*. A subcube-grid is not a mesh-connected processor array. Although a $\gamma_1 \times \gamma_2$ subcube-grid has γ_1 processors in each column and γ_2 processors in each row, the neighboring processors in the grid may or may not be physically connected; instead, each row and each column of processors is required to form a hypercube of smaller dimension, which is a subcube.

We observed in [3] that different communication algorithms may be built on top of the basic subcube-doubling scheme by employing a variety of strategies in *updating* the message to be forwarded to the next neighbor. In particular, we show how *redundant updates* can maintain data proximity so that exactly the same synchronous communication steps may be followed by all processors throughout the computation for all possible choices of the dimensions γ_1 and γ_2. The communication scheme we proposed in [3] allows us to employ the optimal aspect ratio "γ_1/γ_2," which is

*Received by the editors March 14, 1990; accepted for publication (in revised form) December 16, 1991. This work was supported in part by Canadian Natural Sciences and Engineering Research Council grant OGP0008111, by National Aeronautics and Space Administration grant NAGW-1457, and by a research grant from the University of Waterloo.

†Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

‡Present address, Departments of Mathematics and Statistics and Computing and Information Science, University of Guelph, Guelph, Ontario, Canada N1G 2W1.

problem-dependent and is chosen at run time according to the particular dimensions of an input matrix. Since all $(p/2)$ disjoint pairs of processors exchange one message per communication step, the total number of messages is independent of the choice of the aspect ratio. The quantity that the optimal aspect ratio aims at reducing is the message length and hence the total communication volume.

The performance of the parallel QR factorization algorithm described in [3] varied widely, depending upon the choice of the aspect ratio. For some problems, the execution time for the worst aspect ratio was as much as six times greater than for the optimal aspect ratio.

In view of the very substantial *net* saving in execution time and storage usage obtained in our earlier work [3], a natural question to ask is whether the same topology can be adapted to parallelize other numerical algorithms efficiently. In this paper, we propose new subcube-grid algorithms for the efficient parallel implementation of the following numerical algorithms.

1. Gaussian elimination with partial pivoting. In §2, we show not only that its subcube-grid implementation enjoys the same saving as QR factorization, but also that partial pivoting can be incorporated without incurring extra message exchanges.

2. QR factorization with column pivoting. In §3, we show that column pivoting can be incorporated into our parallel algorithms in [3] without increasing the number of messages exchanged. We also show that the increase in computing cost is very small.

3. Multiple least squares updating. While the idea we use to develop the algorithms in [3] can be immediately applied to multiple least squares updating, it is equally important to have an efficient scheme to dynamically relocate the data of the computed Cholesky factor as the aspect ratio of the subcube grid changes. In §4, we show that with an appropriate mapping scheme, we can adapt the subcube doubling technique to obtain a *dynamic* data relocation algorithm for the Cholesky factor, and have thus obtained a feasible parallel algorithm for the continued process of multiple least squares updating.

Timing results for all three algorithms on Intel iPSC/2 and iPSC/860 hypercube multiprocessors are reported in §5, and they are compared with the performance of other known algorithms.

2. Gaussian elimination with partial pivoting. Due to its fundamental and practical role in solving linear systems of equations, Gaussian elimination is often the first numerical algorithm considered when experimenting on a new computer architecture. Since 1985, the year in which hypercube multiprocessors became commercially available, much work has been done on the efficient implementation of this algorithm and its variants on hypercubes [1], [2], [5], [7]–[11], [13]–[16], [22], [23]. The topologies employed include the embedded ring, spanning-tree, and two-dimensional mesh-connected processor array. The communication algorithms devised for these embedded topologies often employ pipelining techniques to reduce communication delay in message passing. The data-mapping strategies considered include column- and row-oriented block or wrap mapping as well as submatrix block mapping. In [22], the theoretical lower bound for communication complexity of the Gaussian elimination (with no pivoting) on a nearest-neighbor mesh network was established to be $O\left(n^2/\sqrt{p}\right) + O\left(n\sqrt{p}\right)$ for a lock-step (synchronous communication) Gaussian elimination algorithm, and $O\left(n^2/\sqrt{p}\right) + O\left(\sqrt{p}\right)$ for any pipelined Gaussian elimination algorithm, where p is the total number of processors. However, we should note that the lower bounds above are obtained assuming that there is *no* startup cost, and that a vector message is pipelined on an element-by-element basis in passing to consecutive processors. These assumptions are not sensible for an iPSC-type hypercube. Optimal concurrent algorithms for Gaussian elimination with and without pivoting are also proposed in [7]. However, since the algorithms proposed in [7] achieve optimality only on a large hypercube machine with little latency and high communication bandwidth, they are not suitable for currently available Intel or NCUBE machines.

While the row or column interchange step is recognized to be crucial in maintaining stable computation in practice, it often causes extra communication and unwelcome disturbance in an otherwise well-pipelined flow of messages. Thus the reduction of the latency caused by pivoting has been the subject of several studies. In [11], Geist and Romine compare the performance of several recently developed pivoting strategies that balance the work load while keeping the communication cost low. They implement all strategies in the C language and report their performance when factoring a matrix of dimension 1024 on an Intel iPSC/1 hypercube of 32 processors. The fastest algorithm is row-oriented, uses a synchronous spanning tree fan-in/fan-out communication algorithm, employs the dynamic pivoting strategy proposed in [11], and applies four pivot rows at a time using a loop unrolling technique. During each of the $(n-1)$ elimination steps, there are two $\log_2 p$ sweeps of the minimum spanning tree rooted at the manager node processor. The two sweeps fan-in/fan-out a short message consisting of the pivot element and the pivot row number. In addition, there is a third $\log_2 p$ sweep of a spanning tree rooted at the node processor that owns the pivot row to be broadcast. Additional communications between possibly nonadjacent nodes are necessary to explicitly permute the pivot row, although they show that the cost is quite modest when the wrap mapping of rows to the p processors need not be maintained. Another topology included in their experiment embeds a ring into the hypercube network, which allows the column-oriented algorithm to run efficiently when the communications can be pipelined. However, since pipelining interferes with loop unrolling, the asynchronous column-oriented algorithm loses to the synchronous row-oriented one on the iPSC/1 hypercube when loop unrolling is allowed.

In this work, we investigate the role of a subcube-grid topology for implementing Gaussian elimination with partial pivoting on hypercubes with substantial message-passing latency. *The algorithm we propose incurs exactly* $\log_2 p$ *synchronous message exchanges between neighboring processors during each elimination step, regardless of whether interchanges occur.* Since the total number of message exchanges is a constant independent of the aspect ratio of the subcube-grid, a square or close-to-square subcube-grid can be employed to reduce the message length and hence the communication volume. Furthermore, since the message length is reduced when the number of processors increases, the subcube-grid algorithms are expected to scale up well.

2.1. A new parallel implementation. We describe the algorithm for factoring a matrix of order n on a hypercube of dimension d. The hypercube is configured as a $\gamma_1 \times \gamma_2$ subcube-grid, where $\gamma_1 = 2^{d_1}$, $\gamma_2 = 2^{d_2}$, and $d_1 + d_2 = d$. The rows of the matrix are wrapped around the γ_1 subcube-rows. Within each subcube-row, the elements are wrapped around the γ_2 processors (that form the subcube) according to their column subscripts. Although our implementation is independent of the aspect values, the chosen d_1 and d_2 should be equal or differ by at most 1 to achieve the optimal performance. The following algorithm is executed by all $p = 2^d$ processors concurrently. The reader should keep in mind that the right-most d_1 bits of the processor *id* correspond to the row mapping, and the left-most d_2 bits correspond to the column mapping.

```
for $i \in [1, n-1]$
    if my share of the matrix contains column $i$ then
        update my message to be my share of column $i$
    else
        label my message content as dummy data
    end if
    $b \leftarrow d_2$ low-order bits of my $id$
    $\ell \leftarrow d_2$
    while $\ell > 0$ do
        send my message to processor with $id$ different
            from my $id$ in bit $b_{\ell-1}$
        receive a message
        if my message contains dummy data then
            update my message to contain the message received
        end if
        $\ell \leftarrow \ell - 1$
    end while
    Store the final content of my message as the pivot column
    Designate the row segment corresponding to the largest element
        in the pivot column as the local pivot row
    if my share of data contains elements from row $i$ and
        the local pivot row number $\neq i$ then
        permute the two segments
    end if
    Update my message to contain the pivot element,
        the local pivot row segment and its row number
    $b \leftarrow d_1$ high-order bits of my $id$
    $\ell \leftarrow d_1$
    while $\ell > 0$ do
        send my message to processor with $id$ different
            from my $id$ in bit $b_{\ell-1}$
        receive a message
        if the magnitude of the pivot element in the message received is
            larger than the magnitude of the pivot element in my message then
            update my message to be the message received
        else if the magnitude of the pivot element in the message received is
            smaller than the magnitude of the pivot element in my message then
            if the message received is marked as row $i$ then
                mark my message as row $i$
                if my message contains my share of data then
                    permute the received message into the local pivot row
                end if
            end if
        else {a tie occurs}
            if the row number in the received message is smaller
                than the row number in my message then
                update my message to be the message received.
            end if
        end if
        $\ell \leftarrow \ell - 1$
    end while
    Use the final content of my message as the pivot row
    if my share of the matrix contains row $i$ and
        the pivot row is different then
        permute my message content into my share of row $i$
    end if
    Use the pivot column and pivot row segments to update my share of data
end for
```

Assuming that $\gamma_1 = \gamma_2 = \sqrt{p}$, the message sizes in the ith reduction step are $\lceil (n-i)/\sqrt{p} \rceil$ for the pivot column and $\lceil n/\sqrt{p} \rceil$ for the pivot row. During each reduction step, the pivot column messages are exchanged between neighboring processors for $\frac{1}{2}\log_2 p$ times, as are the pivot row messages, resulting in a total of exactly $\log_2 p$ synchronous bidirectional exchanges per reduction step. Since the processor that needs to update a message content and the one that needs to perform a local permutation cannot be the same, the *redundant* permutations are *free* concurrent activities.

3. Parallel QR factorization with column pivoting. To incorporate column pivoting into the subcube-grid algorithm in [3], the communication scheme is the same as that proposed in §2 for Gaussian elimination with partial pivoting, apart from applying the redundant (free) permutations to select the pivot column segment instead of the pivot row segment. However, we must note that the criterion for choosing the local pivot column is now different and it uses the "column norm." To avoid communication in updating the column norms after each reduction step, we make use of the important property that the two-norms of vectors are *invariant* under orthogonal transformation. The modification to our original algorithm [3] is quite minor as described below.

First, we compute the initial n column norms and distribute them to the appropriate processors along with the data, i.e., the processor that is assigned data from the ith column will be sent the initial norm of the ith column. Therefore, all of the γ_1 processors in each subcube-column have the initial norms of the same $\lceil n/\gamma_2 \rceil$ columns. This amounts to distributing one extra row segment to each processor, and the cost is not significant if $(m/\gamma_1) \gg 1$, where m is the row dimension of the input matrix.

Second, we observe that since the norm is invariant under orthogonal transformation, the norm of the remaining $(n-i)$ elements in each column can be computed by simply subtracting the norm of the single ith row element from the current norm. This can be accomplished without communication because every processor receives the corresponding segment from the ith row after the last message exchange during the ith reduction step in the original algorithm [3]. Of course, in using this technique, one must avoid loss of accuracy through numerical cancellation. Although our test code does not do this, in a production version some mechanism would have to be incorporated to monitor for this problem, and recompute the column norms when trouble was detected. We do not believe this feature would materially change our timing results or our conclusions.

We can thus conclude that the communication complexity remains the same order as our results in [3], and that the extra computing cost incurred in updating the norms is evidently small.

4. Multiple least squares updating. The implementation of the *multiple* least squares updating algorithm on the hypercube was first studied by Kim, Agarwal, and

Plemmons in [18]. The proposed algorithms in [18] wrap mapped the Cholesky factor and the incoming rows to the p processors either following the processor *ids* or the processor order of an embedded linear array. Thus the communication cost in updating an $n \times n$ Cholesky factor by annihilating the input $m \times n$ matrix is $O\left(n^2 \log_2 p\right)$ in either case. We obtain the same result in [3] when employing the hypercube as a $p \times 1$ subcube-grid in reducing an $m \times n$ matrix to upper triangular form via orthogonal transformation. Although the communication cost is *independent* of the row dimension m of the input matrix and this configuration of the subgrid would be the optimal choice if $m \gg n$, it is not the desirable algorithm otherwise. Due to the usually stringent time constraint in engineering applications, it is rare for $m \gg n$ when the Cholesky factor must be updated by the newly available data. Thus it is very important that the parallel implementation be flexible and efficient for all cases. Since the algorithm we proposed in [3] was designed to work equally well for matrices of all possible dimensions, it is certainly a good candidate for multiple least squares updating on the hypercube. We show here that by using a different data-mapping strategy for the computed Cholesky factor, we obtain a parallel algorithm that not only always chooses the optimal aspect ratio for the embedded subcube-grid according to the row and column dimensions of the newly available data, but also allows *dynamic* relocation of the Cholesky factor if the aspect ratio changes. We also show that the new data-mapping strategy maintains the work load as balanced as before.

Let us express the multiple least squares updating problem by

$$\begin{pmatrix} R_{n\times n} \\ A_{m\times n} \end{pmatrix} \longrightarrow \begin{pmatrix} \tilde{R}_{n\times n} \\ 0 \end{pmatrix},$$

where $\tilde{R}$ results from modifying the upper triangular factor R by zeroing out all elements in the $m \times n$ matrix A via orthogonal transformations. We describe the new data-mapping strategy and the dynamic data relocation scheme in the following sections.

4.1. New data-mapping strategy. Instead of wrap mapping the rows and columns of $R_{n\times n}$ and $A_{m\times n}$ to the $\gamma_1 \times \gamma_2$ subcube-grid, we propose *block* mapping for the rows while maintaining wrap mapping for the columns. That is, each subcube-row of processors contains a block of $\lceil n/\gamma_1 \rceil$ consecutive rows from $R_{n\times n}$ and a block of $\lceil m/\gamma_1 \rceil$ rows from $A_{m\times n}$. With each subcube-row, the data from each block are wrap mapped to the γ_2 processors in the subcube according to their column subscripts, exactly as proposed in [3].

We want to emphasize here that the wrap mapping of rows was necessary to maintain balanced work load in reducing a full $m \times n$ matrix to its upper triangular form, but it is no longer necessary if the entire matrix A is zeroed out by updating an available triangular R. The reasons are two-fold:

1. In the latter case, all elements in the same column of $A_{m\times n}$ are modified the same number of times regardless of the row number.

2. The subcube-doubling communication algorithm as adapted in [3] has the unique feature that all processors always perform $\log_2 p$ synchronous exchanges of messages regardless of the location of the pivot row. Consequently, the fact that each processor owns consecutive pivot rows from R will not affect the work load distribution at all.

4.2. A dynamic parallel data relocation scheme. Since the row dimension of the data matrix A varies with the amount of data available at different times, the subcube-grid is to be dynamically configured to achieve the optimal aspect ratio and the new input matrix will be distributed to the processors accordingly.

When the aspect ratio differs from the one used in the previous update, it becomes necessary to *relocate* the previously distributed Cholesky factor R. The mapping strategy we proposed above allows us to relocate R to the new $\tilde{\gamma}_1 \times \tilde{\gamma}_2$ subcube-grid efficiently. The following observations are the keys to the design of the parallel algorithm.

1. $\tilde{\gamma}_1 = 2^{\tilde{d}_1}$ and $\tilde{\gamma}_2 = 2^{\tilde{d}_2}$, where $\tilde{d}_1 + \tilde{d}_2 = d$, and d is the dimension of the hypercube.

2. $\tilde{\gamma}_1 = \gamma_1 \times 2^i$ and $\tilde{\gamma}_2 = \gamma_2 \times 2^{-i}$, for $-d_1 \le i \le d_2$. By this observation, we simply need an algorithm for $i = 1$, because the same algorithm can be executed $i > 1$ times to get the resulting distribution. This observation contributes to the simplicity of the algorithm.

3. The sequential ordering of processors on the subcube-grid following the processor *id* row by row plays an important role in the parallel algorithm.

4.2.1. The parallel algorithm. Since the row dimension of the input matrix is initially known to the host, we require the host to compute the new aspect $\tilde{\gamma}_1$ and send it to all p node processors in the hypercube together with the data from the input matrix A. Therefore, there is *no* extra communication involved in broadcasting the value $\tilde{\gamma}_1$ to all processors in the subcube-grid, and $\tilde{\gamma}_2 = p/\tilde{\gamma}_1$ can then be computed by each processor. The parallel algorithm can be best explained by an example. We consider the case of changing a 4×4 subcube-grid to a 2×8 subcube-grid as shown below.

$$\begin{array}{llll} P_0 & P_1 & P_2 & P_3 \\ P_4 & P_5 & P_6 & P_7 \\ P_8 & P_9 & P_{10} & P_{11} \\ P_{12} & P_{13} & P_{14} & P_{15} \end{array} \longrightarrow \begin{array}{llllllll} P_0 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 \\ P_8 & P_9 & P_{10} & P_{11} & P_{12} & P_{13} & P_{14} & P_{15} \end{array}$$

Since the columns of the factor R were wrap mapped to the processors in each subcube row, processors P_0, P_1, P_2, and P_3 can simultaneously send half of the data (corresponding to appropriate columns) to P_4, P_5, P_6, and P_7. Since a block of consecutive rows are stored in each processor, processors P_4, P_5, P_6, and P_7 will send one-half of their data (corresponding to appropriate columns) to processors P_0, P_1, P_2, and P_3 at the same time. The processors P_8, P_9, P_{10}, and P_{11} will understandably exchange the same amount of data with processors P_{12}, P_{13}, P_{14}, and P_{15} at the same time. Again, because of the subcube-grid connectivity, this amounts to "one" synchronous message exchange between a pair of directly connected processors. Each message consists of half of the data the processor previously has and thus data distribution remains balanced.

4.2.2. Communication complexity results. We can thus conclude that for $i = 1$, the communication cost is one near-neighbor exchange of one message of $(n^2/2p)$ floating-point numbers. Since $i \le d$, the communication volume for relocating R is $\left((n^2 \log_2 p)/2p\right)$ in the worst case.

5. Numerical experiments on an iPSC/2 and an iPSC/860.

In this section we report timing results on a 64-node Intel iPSC/2 hypercube and a 128-node iPSC/860 hypercube at Oak Ridge National Laboratory. The ratio of communication speed (measured by eight-byte one-hop transfer time λ) to computation speed (measured by eight-byte multiply time τ) is $\lambda/\tau \approx 59$ for the iPSC/2 and $\lambda/\tau \approx 1000$ for the iPSC/860 [6]. Our analysis in [3] suggests that the optimal aspect ratio of the subcube-grid is insensitive to such a hardware characteristic. Our conclusion is supported by the consistent performance of our various subcube-grid algorithms on iPSC/1 [3], iPSC/2, and iPSC/860. Our algorithms are implemented in double-precision Fortran 77. We have two implementations for the QR factorization algorithm. The basic version uses Givens rotation and an enhanced version uses a hybrid scheme [21] adapted for subcube-grid implementation in [3]. The QR factorization times represent the basic Givens version unless they are specifically indicated as being from the hybrid version. Our Fortran programs are compiled using the Green Hill Fortran compiler (v1.8.5). The compile time option is specified in each table.

5.1. Sequential times. In Table 1 we report the sequential execution times for both Gaussian elimination with partial pivoting, T_{GEPP}, and QR factorization with column pivoting, T_{QRCP}. Since our parallel code reduces to a sequential code with negligible overhead when running on a 1×1 subcube- grid, the sequential times are obtained from running the parallel code on a single node on the iPSC/2 or the iPSC/860 machine.

TABLE 1
Sequential times on an iPSC/2 *and an* iPSC/860.

Sequential times on an iPSC/2
(f77 compile time option: -OLM)

$m = n$	T_{GEPP} (sec)	T_{QRCP} (sec)
100	4.7	13.5
200	37.4	102.6
300	125.4	340.3
400	296.1	799.3
500	577.0	1552.9

Sequential times on an iPSC/860
(f77 compile time option: - OLM)

$m = n$	T_{GEPP} (sec)	T_{QRCP} (sec)
100	0.2	0.76
200	1.4	4.4
300	4.8	13.1
400	11.3	28.7
500	22.1	53.6

For larger matrices, the estimates for the sequential times were obtained from the data in Table 1 through a least squares fit of the polynomial representing the analytical expression for the execution time. Listed in Table 2 are the estimated times for matrices of dimensions 1000, 1024, 2000, and 2048. These values are useful in estimating the speedup and efficiency of the parallel algorithms.

5.2. Parallel times and performance comparison with other algorithms.

5.2.1. Gaussian elimination with partial pivoting. Efficient C programs for parallel Gaussian elimination with partial pivoting as proposed by Geist and Romine [11] were provided by the authors in the appendix of report [12]. Since the C programs in [12] were originally coded for an iPSC/1, the older first generation of hypercube, we have made necessary changes to call the iPSC/2 and iPSC/860 message-passing primitives. In addition, we have also converted the C code from the original single-precision mode to double-precision mode so that their performance can be compared with our algorithms, which are coded in double-precision Fortran.

TABLE 2
Least squares estimates on an iPSC/2 *and an* iPSC/860.

Least squares estimates on an iPSC/2		
$m = n$	T_{GEPP} (sec)	T_{QRCP} (sec)
1000	4,595	12,293
1024	4,933	13,196
2000	36,678	97,833
2048	39,381	105,034

Least squares estimates on an iPSC/860		
$m = n$	T_{GEPP} (sec)	T_{QRCP} (sec)
1000	176	388
1024	189	416
2000	1,406	2,944
2048	1,510	3,157

The test matrices we use for comparison are of dimensions 1024, 2048, and 4096 for $p = 32$, 64, and 128, because the code in [12] does not run if the matrix dimension n is not an integral multiple of p, the number of processors.

For the RSRP algorithm [11], [12], the effectiveness of the loop unrolling technique is measured by applying i pivot rows at a time (nmod=i, $i = 1, 2, 3, 4$). The pipelined CSRP algorithm was originally coded for applying two pivot columns at a time (nmod=2) [11], [12]. The timing results in Table 3 indicate that the technique of loop unrolling appears to be much less effective on an iPSC/860 than on an iPSC/2. Second, the results in Table 3 indicate that the pipelined CSRP algorithm performs significantly better than RSRP for both $p = 32$ and $p = 64$ processors on an iPSC/860, while CSRP performs better than RSRP on an iPSC/2 with 64 processors. Third, for a 1024 × 1024 matrix, there is little improvement to the iPSC/860 execution time by doubling the number of processors from 32 to 64 due to the dominating communication cost.

TABLE 3
Execution times of C code by Geist/Romine [12].

Gaussian elimination with partial pivoting						
Execution times of C code by Geist/Romine [12] (modified for iPSC/2 in double-precision) cc compile time option: –OLM						
n	p	RSRP				pipelined CSRP
		nmod = 1	nmod = 2	nmod = 3	nmod = 4	nmod = 2
1024	32	172.5 sec	139.4 sec	136.1 sec	133.8 sec	139.0 sec
	64	99.3 sec	84.2 sec	84.3 sec	84.9 sec	79.1 sec
Execution times of C code by Geist/Romine [12] (modified for iPSC/860 in double-precision) cc compile time option: –OLM						
n	p	RSRP				pipelined CSRP
		nmod=1	nmod=2	nmod=3	nmod=4	nmod=2
1024	32	17.7 sec	16.7 sec	17.8 sec	17.8 sec	10.3 sec
	64	16.5 sec	16.1 sec	16.7 sec	16.8 sec	7.7 sec

Table 4 compares the double-precision C implementation of the RSRP and the pipelined CSRP algorithms [11], [12] with our Fortran implementation of Gaussian elimination with partial pivoting. Our Fortran times were obtained when the optimal aspect ratio was used to configure the subcube-grid. Some RSRP times are missing from Table 4 because the factorization was not completed in a reasonable length of time. Our correspondence with the author indicates that the execution times were probably prolonged by calls to trap handlers such as underflow.

TABLE 4
Comparing different Gaussian elimination algorithms on an iPSC/2 *and an* iPSC/860.

Gaussian elimination with partial pivoting Double-precision C/FORTRAN times on an iPSC/2				
n	p	RSRP (nmod=1) (cc -OLM)	pipelined CSRP (nmod=2) (cc -OLM)	Subcube-grid (nmod=1) (f77 -OLM)
1024	32	172.5 sec	139.0 sec	171.9 sec
	64	99.3 sec	79.1 sec	94.7 sec

Gaussian elimination with partial pivoting Double-precision C/FORTRAN times on an iPSC/860				
n	p	RSRP (nmod=1) (cc -OLM)	pipelined CSRP (nmod=2) (cc -OLM)	Subcube-grid (nmod=1) (f77 -OLM)
1024	32	17.7 sec	10.3 sec	11.6 sec
	64	16.5 sec	7.7 sec	9.1 sec
2048	32		63.1 sec	68.4 sec
	64		41.2 sec	40.1 sec
	128		30.3 sec	31.3 sec
4096	64		252.0 sec	240.9 sec
	128		164.2 sec	152.8 sec

5.2.2. QR factorization with column pivoting. In Table 5 we report the iPSC/2 and iPSC/860 times obtained from running our parallel QR factorization algorithm with and without pivoting on the optimal subcube-grid configuration.

The most current QR factorization with column pivoting times was reported by Coleman and Plassmann [4], [19], [20] and is quoted in Table 6. The algorithm proposed in [4] and [19] wrap maps the rows of an $m \times n$ matrix $(m \geq n)$ to a ring of processors embedded in a hypercube network. Comparing the performance of our subcube-grid (Givens) algorithm with that of the Householder algorithm implemented on a ring of p processors [19] in Table 6, we observe that the saving in communication time by employing an optimal subcube-grid enables the Givens algorithm to be competitive in the cases $m = n = 400$ $(p = 16, 32)$; $m = 400$, $n = 200$ $(p = 32)$; $m = 400$, $n = 100$ $(p = 32)$; and $m = 200$, $n = 100$ $(p = 16, 32)$. The actual differences in the execution times appear to be smaller when the compile time option –OLM is used. In the case of $m = n = 400$ and $p = 16$, the subcube-grid Givens algorithm is slightly faster than the Householder algorithm without the –OLM option (68.3 seconds versus 70.7 seconds), but the situation reverses when the –OLM option is used (58.2 seconds versus 53.6 seconds).

In general, we note that the differences between the f77 times and f77 –OLM times appear to be consistently more drastic in Plassmann's Householder algorithm: the reduction in times ranges from 28 percent to 31 percent in most cases. This is contrary to the 14 percent to 20 percent reduction in the subcube-grid Givens algorithm and the 10 percent to 15 percent reduction in the subcube-grid hybrid algorithm.

Since Plassmann's algorithm employs row-oriented Householder transformations, the timing results reported in [19] are also compared with a *hybrid* version of our

TABLE 5
Parallel QR *factorization on an* iPSC/2 *and an* iPSC/860.

QR factorization (by Givens rotations)
Parallel times on an iPSC/2
(f77 compile time option: –OLM)

m	n	p	T_{QR} (sec) (no pivoting)	T_{QRCP} (sec) (with pivoting)	Optimal $\gamma_1 \times \gamma_2$
1000	1000	32	406.7	421.8	8×4
		64	215.7	223.8	8×8
1200	800	32	367.1	377.7	8×4
		64	191.6	201.9	8×8
800	1200	32	363.5	376.7	4×8
		64	196.5	199.4	8×8
3000	300	32	163.3	167.0	32×1
		64	87.3	87.4	32×2
300	3000	32	157.1	162.7	2×16
		64	84.1	87.6	2×32

QR factorization (by Givens rotations)
Parallel times on an iPSC/860
(f77 compile time option: –OLM)

m	n	p	T_{QR} (sec) (no pivoting)	T_{QRCP} (sec) (with pivoting)	Optimal $\gamma_1 \times \gamma_2$
1000	1000	32	19.4	23.0	8×4
1200	800	32	18.2	20.9	8×4
800	1200	32	17.9	21.4	4×8
3000	300	32	8.1	8.9	32×1
300	3000	32	7.5	9.1	2×16

subcube-grid algorithm. The hybrid version combines Householder transformations and Givens rotations. It involves slightly more arithmetic operations than using Householder transformations alone, but it is particularly suitable for the subcube-grid implementation [3]. This idea of using both types of transformations in a somewhat different context was proposed by Pothen and Raghavan in [21]. The results in Table 6 indicate the significant saving from running the hybrid algorithm on the optimal subcube-grid. In addition, while Plassmann's times include the transformation of right-hand side [4], [19], [20], our times include extra arithmetic due to the scaling operations employed in our implementations to avoid overflow and underflow. Finally, none of the sequential or parallel implementations use LINPACK BLAS. The results in Table 6 again indicate the influence of compile time option –OLM. While compiling the code without the –OLM option, running the hybrid algorithm on the optimal subcube-grid gives the minimum execution times. With the –OLM option, we note that the QRFAC routine from MINPACK [17] gives the fastest sequential times after a reduction of 28–29 percent. Comparing the parallel Householder algorithm with the subcube-grid hybrid algorithm, the timing results are comparable for cases when $m \gg n$ and the optimal $\gamma_1 \times \gamma_2$ is given by $p \times 1$ or $(p/2) \times 2$. The saving in communication time by employing the optimal subcube-grid becomes evident in cases $m = n = 400$ ($p = 16, 32$), $m = 400$, $n = 200$ ($p = 32$), $m = 200$, $n = 100$ ($p = 16, 32$). More significant saving in execution times can thus be expected from the subcube-grid hybrid algorithm for cases $m < n$.

5.2.3. Multiple least squares updating. In Table 7 we report the times for updating an $n \times n$ Cholesky factor by an $m \times n$ matrix, and the worst possible relocation

TABLE 6
Comparing different QR *factorization algorithms on an* iPSC/2.

QR factorization with column pivoting Double-precision FORTRAN times (seconds) on an iPSC/2									
			Plassmann [19] (Householder)		Subcube-grid (Givens⋆⋆)		Subcube-grid (hybrid⋆⋆)		Optimal
m	n	p	f77 [19]	f77 -OLM [20]	f77	f77 -OLM	f77	f77 -OLM	$\gamma_1 \times \gamma_2$
400	400	1	‡785.6	‡560.3†	958.6	799.3	690.0*	590.9	1×1
		8	119.1	85.2	128.8	108.7	93.2*	82.0†	4×2
		16	70.7	53.6	68.3	58.2	49.3*	43.6†	4×4
		32	47.5	40.3	37.5	32.4	28.8*	26.0†	8×4
800	200	1	‡547.8	‡390.5†	674.1	564.2	485.4*	412.1	1×1
		8	75.2	53.3†	90.1	74.8	62.9*	55.3	4×2
		16	40.6	28.9†	46.7	39.9	33.1*	29.3	8×2
		32	23.8	16.9	25.1	21.7	18.7*	16.7†	16×2
400	200	1	‡250.0	‡178.5†	306.5	256.6	219.4*	188.2	1×1
		8	36.9	26.5	41.9	35.7	29.7*	26.2†	4×2
		16	21.7	15.5	22.6	19.4	16.8*	15.0†	8×2
		32	14.4	11.7	13.1	11.3	9.8*	8.8†	8×4
1600	100	1	‡303.1	‡216.1†	376.2	317.2	267.3*	230.0	1×1
		8	40.1	27.8†	48.5	41.0	33.8*	29.7	8×1
		16	21.0	14.9†	25.0	21.4	17.8*	15.7	16×1
		32	11.6	8.2†	13.5	11.6	10.0*	9.0	32×1
800	100	1	‡147.1	‡105.6†	184.1	155.2	131.5*	112.7	1×1
		8	20.1	14.1†	24.3	20.6	17.1*	15.1	8×1
		16	11.0	7.9†	13.0	11.2	9.6*	8.5	16×1
		32	6.6	5.0†	7.4	6.4	5.5*	5.0†	16×2
400	100	1	‡70.6	‡50.8†	88.1	74.3	63.5*	54.1	1×1
		8	10.4	7.6†	12.2	10.5	9.0*	8.0	8×1
		16	6.2	4.8	6.9	6.0	5.1*	4.6†	8×2
		32	4.2	3.4	4.1	3.6	3.4*	3.0†	16×2
200	100	1	‡32.4	‡23.3†	40.0	33.8	28.7*	24.8	1×1
		8	5.6	4.3	6.1	5.3	4.6*	4.1†	4×2
		16	3.8	3.1	3.8	3.2	2.9*	2.7†	4×4
		32	3.0	2.7	2.5	2.2	2.0*	2.0†	8×4

*The minimum f77 execution time.
†The minimum f77 -OLM execution time.
‡The QRFAC routine from MINPACK [17] reported in [19] and [20].
⋆⋆Scaling is employed to avoid overflow/underflow in this implementation.

cost. Our experiments show tremendous saving by relocating the Cholesky factor in all cases. As noted in Table 7, Givens rotations were employed in our algorithm to update the Cholesky factor. The updating time can therefore be further reduced by using the hybrid QR factorization scheme.

Acknowledgments. We thank Oak Ridge National Laboratory for the use of their iPSC/2 and iPSC/860 hypercube multiprocessors, and we would like to acknowledge programming assistance by the following students: Tom Abbott, Willis Constantineau, Phil Edmonds, and Darcy Quesnel.

TABLE 7
Multiple least squares updating times on an iPSC/2 *and an* iPSC/860.

Multiple least squares updating (by Givens rotations)
Double-precision FORTRAN times on an iPSC/2
(f77 compile time option: None)

$m = 1000$, $n = 1000$, $p = 64$

From	To	Relocation	Updating	Total
$\gamma_1 \times \gamma_2$	$\gamma_1 \times \gamma_2$	(sec)	(sec)	(sec)
64×1	64×1	0.012	431.5	431.5
	32×2	2.689	379.1	381.8
	16×4	3.189	363.1	366.3*
	8×8	3.307	367.9	371.2

$m = 100$, $n = 2000$, $p = 64$

From	To	Relocation	Updating	Total
$\gamma_1 \times \gamma_2$	$\gamma_1 \times \gamma_2$	(sec)	(sec)	(sec)
64×1	64×1	0.023	531.5	531.5
	16×4	4.690	214.2	218.9
	2×32	9.039	168.2	177.2*
	1×64	10.574	191.1	201.7

$m = 2000$, $n = 100$, $p = 64$

From	To	Relocation	Updating	Total
$\gamma_1 \times \gamma_2$	$\gamma_1 \times \gamma_2$	(sec)	(sec)	(sec)
1×64	1×64	0.005	60.9	60.9
	16×4	2.583	10.0	12.6
	32×2	2.889	8.9	11.8
	64×1	2.987	8.7	11.7*

Multiple least squares updating (by Givens rotations)
Double-precision FORTRAN times on an iPSC/860
(f77 Compile time option: None)

$m = 2000$, $n = 2000$, $p = 64$

From	To	Relocation	Updating	Total
$\gamma_1 \times \gamma_2$	$\gamma_1 \times \gamma_2$	(sec)	(sec)	(sec)
64×1	64×1	0.005	143.5	143.5
	32×2	0.572	111.6	112.2
	16×4	0.864	103.1	104.0*
	8×8	1.340	113.7	115.0

$m = 1000$, $n = 1500$, $p = 64$

From	To	Relocation	Updating	Total
$\gamma_1 \times \gamma_2$	$\gamma_1 \times \gamma_2$	(sec)	(sec)	(sec)
1×64	1×64	0.004	155.8	155.8
	2×32	0.302	83.5	83.8
	4×16	0.716	51.8	52.5
	8×8	1.166	38.8	40.0*

$m = 100$, $n = 2000$, $p = 64$

From	To	Relocation	Updating	Total
$\gamma_1 \times \gamma_2$	$\gamma_1 \times \gamma_2$	(sec)	(sec)	(sec)
64×1	64×1	0.005	69.7	69.7
	16×4	0.918	20.3	21.2
	4×16	1.939	13.1	15.0*
	1×64	2.755	23.6	26.4

*Minimum execution time.

REFERENCES

[1] R. M. Chamberlain, *An algorithm for* LU *factorization with partial pivoting on the hypercube*, in Hypercube Multiprocessors Proceedings 1987, Michael T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[2] E. C. H. Chu and J. A. George, *Gaussian elimination with partial pivoting and load balancing on a multiprocessor*, Parallel Comput., 5 (1987), pp. 65–74.

[3] ———, QR *factorization of a dense matrix on a hypercube multiprocessor*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 990–1028.

[4] T. F. Coleman and P. Plassmann, *Solution of nonlinear least-squares problems on a multiprocessor*, Tech. Rep. CS-88-923, Computer Science Dept., Cornell Univ., Ithaca, NY, 1988.

[5] G. J. Davis, *Column* LU *factorization with pivoting on a message-passing multiprocessor*, SIAM J. Algebraic Discrete Meth., 7 (1986), pp. 538–550.

[6] T. Dunigan, *Performance of the Intel* iPSC/860 *hypercube*, Tech. Rep. ORNL/TM-11491, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, June 1990.

[7] G. Fox, W. Furmanski, and D. Walker, *Optimal matrix algorithm on homogeneous hypercubes*, in Third Conf. Hypercube Concurrent Computers and Applications, Association for Computing Machinery, 1988, pp. 1656–1673.

[8] G. A. Geist, *Efficient parallel* LU *factorization with pivoting on a hypercube multiprocessor*, Tech. Rep. ORNL-6290, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[9] G. A. Geist and M. T. Heath, *Parallel Cholesky factorization on a hypercube multiprocessor*, Tech. Rep. ORNL-6211, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[10] ———, *Matrix factorization on a hypercube multiprocessor*, in Hypercube Multiprocessors, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadephia, PA, 1986.

[11] G. A. Geist and C. H. Romine, LU *factorization algorithms on distributed-memory architectures*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 639–649.

[12] ———, LU *factorization algorithms on distributed-memory architectures*, Tech. Rep. ORNL/TM-11491, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, March 1987.

[13] A. Gerasoulis, N. Missirlis, I. Nelken, and R. Peskin, *Implementing Gauss Jordan on a hypercube multiprocessor*, in Third Conf. Hypercube Concurrent Computers and Applications, Association for Computing Machinery, 1988, pp. 1569–1576.

[14] M. T. Heath, *Parallel Cholesky factorization in message passing multiprocessor environments*, Tech. Rep. ORNL-6150, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[15] S. L. Johnsson, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distrib. Comput., 4 (1987), pp. 133–172.

[16] C. Moler, *Matrix computation on distributed memory multiprocessors*, in Hypercube Multiprocessors 1986, M.T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[17] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *User guide for minpack*-1, Tech. Rep. ANL-80-74, Argonne National Laboratory, Argonne, IL, 1980.

[18] S. Kim, D. P. Agarwal, and R. J. Plemmons, *Recursive least squares filtering for signal processing on distributed memory multiprocessors*, Tech. Rep. draft, Dept. of Electrical and Computer Engineering and Dept. of Computer Science and Mathematics, North Carolina State Univ., Raleigh, NC, March 1988.

[19] P. E. Plassmann, *The parallel solution of nonlinear least-squares problems*, Ph.D. thesis, Cornell University, Ithaca, NY, Jan. 1990.

[20] ———, Personal communication, Feb. 1991.

[21] A. Pothen and P. Raghavan, *Distributed orthogonal factorization: Givens and Householder algorithms*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1113–1134.

[22] Y. Saad, *Communication complexity of the Gaussian elimination algorithm on multiprocessors*, Linear Algebra Appl., 77 (1986), pp. 315–340.

[23] D. W. Walker, T. Aldcroft, A. Cisneros, G. C. Fox, and W. Furmanski, LU *decomposition of banded matrices and the solution of linear systems on hypercubes*, in Third Conf. Hypercube Concurrent Computers and Applications, Association for Computing Machinery, 1988, pp. 1635–1655.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 95–120, January 1993

STABILITY OF COMPUTATIONAL METHODS FOR CONSTRAINED DYNAMICS SYSTEMS*

URI M. ASCHER† AND LINDA R. PETZOLD‡

Abstract. Many methods have been proposed for numerically integrating the differential-algebraic systems arising from the Euler–Lagrange equations for constrained motion. These are based on various problem formulations and discretizations. We offer a critical evaluation of these methods from the standpoint of stability.

Considering a linear model, we first give conditions under which the differential-algebraic problem is well conditioned. This involves the concept of an essential underlying ODE. We review a variety of reformulations which have been proposed in the literature and show that most of them preserve the stability of the original problem. Then we consider stiff and nonstiff discretizations of such reformulated models. In some cases, the same implicit discretization may behave in a very different way when applied to different problem formulations, acting as a stiff integrator on some formulations and as a nonstiff integrator on others. We present the approach of projected invariants as a method for yielding problem reformulations which are desirable in this sense.

Key words. differential-algebraic equations, Euler–Lagrange equations, stability, multibody systems, numerical ODEs

AMS(MOS) subject classifications. 65L05, 70H35

1. Introduction.

Various techniques have been proposed in the literature for the numerical solution of the Euler–Lagrange equations, which govern the motion of mechanical systems with constraints [19]. Several of these techniques are used in commercial codes. The equations to be solved form a system of second-order ordinary differential equations (ODEs) for the (generalized) multibody coordinates. They also involve Lagrange multiplier functions and are subject to constraints, e.g., on configuration and/or motion. Mathematically, this may be considered as a system of differential-algebraic equations (DAEs) of index 3 in a special semi-explicit form [6]. It is well known that a direct discretization of such a DAE yields numerical difficulties; this is what gives rise to a multitude of other, more specific solution techniques.

Typically, such a solution technique consists of a step of problem reformulation, which involves reducing its index, followed by a discretization of the resulting formulation. In recent work [9], [10] it has been shown that for a certain model problem, some of these formulations can be equivalent. An important consideration in selecting an appropriate solution method (i.e., a combination of formulation and discretization) is the stability of the method and the subsequent stability restrictions that a chosen step size must satisfy. In this paper we investigate the stability of various solution techniques.

In order to be more specific, we write the Euler–Lagrange equations for a constrained multibody system

$$M(\mathbf{p})\mathbf{p}'' = \mathbf{f}(\mathbf{p},\mathbf{v}) - G^T(\mathbf{p})\lambda - \hat{G}^T(\mathbf{p})\hat{\lambda}, \tag{1.1a}$$

*Received by the editors April 29, 1991; accepted for publication (in revised form) December 10, 1991.

†Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z2 (ascher@cs.ubc.ca). The work of this author was partially supported under Natural Sciences and Engineering Research Council of Canada grant OGP0004306.

‡Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455 (petzold@cs.umn.edu). The work of this author was partially supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, and by Lawrence Livermore National Laboratory contract W-7405-Eng-48.

$$\text{(1.1b)} \qquad \mathbf{0} = \mathbf{g}(\mathbf{p}),$$
$$\text{(1.1c)} \qquad \mathbf{0} = \hat{G}(\mathbf{p})\mathbf{v} + \hat{\mathbf{g}}(\mathbf{p}).$$

Here the unknowns are: $\mathbf{p}$, the generalized coordinates; $\mathbf{v} \equiv (d\mathbf{p}/dt) \equiv \mathbf{p}'$, the generalized velocities; and λ and $\hat{\lambda}$, the Lagrange multiplier functions. In (1.1a) M is the mass matrix (we consider such formulations where $M(\mathbf{p})(t) \in \mathcal{R}^{n_p \times n_p}$ is symmetric positive definite), $\mathbf{f}$ stands for the applied forces, and $G(\mathbf{p})$ is the Jacobian matrix of the holonomic constraints

$$\text{(1.2)} \qquad G(\mathbf{p}) = \mathbf{g}_\mathbf{p}, \qquad G(\mathbf{p})(t) \in \mathcal{R}^{n_\lambda \times n_p}.$$

Similarly, $\hat{G} = \hat{\mathbf{g}}_\mathbf{p}, \hat{G}(\mathbf{p})(t) \in \mathcal{R}^{n_{\hat{\lambda}} \times n_p}$, and we assume that the matrix $\begin{pmatrix} G^T & \hat{G}^T \end{pmatrix}$ has a full column rank (i.e., the constraints are independent) for each t.

In (1.1b) there are n_λ configuration (position) constraints and in (1.1c) there are $n_{\hat{\lambda}}$ motion or other constraints. For simplicity of presentation we shall often assume that either $n_{\hat{\lambda}} = 0$ (hence $\hat{\lambda}$ disappears from (1.1a) as well), i.e., that there are only holonomic constraints, or that $n_\lambda = 0$ (whence λ disappears from (1.1a)).

Clearly, two differentiations of the constraints (1.1b) allow elimination of λ from (1.1a). Thus the original DAE has index 3. On the other hand, if $n_\lambda = 0$ and $n_{\hat{\lambda}} > 0$, then only one differentiation of (1.1c) is needed to eliminate $\hat{\lambda}$ and obtain an ODE, so the index is 2. Both of these cases can be cast in the form (2.1) below with $m = 2$ and $m = 1$ (for the equivalent first-order form of (1.1a)), respectively.

In order to give a methodical stability discussion we proceed in stages and consider the linearized form of the DAE (1.1). The class of nonlinear problems considered here behaves like its linear variational form away from singularities (i.e., in a neighborhood of an isolated solution). Thus our arguments will be valid in these general circumstances. We assume that the given linear DAE problem is well conditioned, and in §2 specify precisely what this means using a constructed *essential underlying ODE* (EUODE). The theory includes the linearizations of (1.1) as special cases.

In §3 we then consider a variety of problem reformulations and show that they also are well conditioned under certain reasonable assumptions. We cover the Baumgarte stabilization technique, a variety of "stabilized" and direct index reductions, and transformations to state-space form. This allows us to consider in §4 discretizations of the various formulations.

We consider stiff and nonstiff discretizations of such reformulated models. In some cases, the same backward differentiation formula (BDF) discretization (see, e.g., [6]), or other stiff discretizations, may behave in a very different way when applied to different reformulations of the same problem, acting as a stiff integrator on some formulations and as a nonstiff integrator on others. The need to restrict the step size in BDF for numerical stability occasionally arises even in formulations that explicitly enforce the constraints. For (1.1), assuming that there are only position constraints which vary on the scale of the solution, such a situation may arise if the mass matrix M has both large and moderate eigenvalues, in which case $M^{-1}G^T$ may be much less pleasant than G. (A corresponding physical situation is a heterogeneous multibody system, i.e., a system that includes bodies with very different masses.[1]) We present the approach of *projected invariants* with a particular choice of the projection as a method for yielding problem reformulations that are desirable in this sense. Section 5 concludes with a summary and recommendations based on our results.

[1]We thank Dr. Dan Rosenthal of RASNA Corporation for illuminating us on this point.

Throughout this paper, we use the following notation: Let $|\cdot|$ be the Euclidean vector norm. For a matrix A we denote the induced matrix norm by $\|A\|$. For a function $\mathbf{u}(t)$, $0 \le t \le t_f$, we denote the corresponding max function norm by $\|\mathbf{u}\| := \max\{|\mathbf{u}(t)|, 0 \le t \le t_f\}$.

2. Problem conditioning.

The DAE of order m,

$$\mathbf{x}^{(m)} = \mathbf{f}(\mathbf{z}(\mathbf{x}), \mathbf{y}, t), \tag{2.1a}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{x}, t), \tag{2.1b}$$

where $\mathbf{z}_j(t) \equiv \mathbf{x}^{(j-1)}(t) := (d^{j-1}\mathbf{x}(t)/dt^{j-1})\,(1 \le j \le m)$ and

$$\mathbf{z}(\mathbf{x})(t) = (\mathbf{x}^T(t), (\mathbf{x}')^T(t), \ldots, (\mathbf{x}^{(m-1)})^T(t))^T, \tag{2.2}$$

has index $m+1$ if $\mathbf{g_x f_y}$ is nonsingular for all t, $0 \le t \le t_f$. The Euler–Lagrange equations for dynamical systems with holonomic constraints are in this form with $m = 2$, $\mathbf{x}$ the generalized coordinates, and $\mathbf{y}$ the Lagrange multipliers. Here we consider the linear (or linearized) form

$$\mathbf{x}^{(m)} = \sum_{j=1}^{m} A_j \mathbf{z}_j + B\mathbf{y} + \mathbf{q}, \tag{2.3a}$$

$$\mathbf{0} = C\mathbf{x} + \mathbf{r}, \tag{2.3b}$$

where A_j, B, and C are smooth functions of t, $0 \le t \le t_f$, $A_j(t) \in \mathcal{R}^{n_x \times n_x}$, $j = 1, \ldots, m$, $B(t) \in \mathcal{R}^{n_x \times n_y}$, $C(t) \in \mathcal{R}^{n_y \times n_x}$, $n_y \le n_x$, and CB is nonsingular for each t (hence the DAE has index $m+1$). All matrices involved, together with their derivatives, are assumed to be uniformly bounded in norm by a constant of moderate size. The inhomogeneities are $\mathbf{q}(t) \in \mathcal{R}^{n_x}$ and $\mathbf{r}(t) \in \mathcal{R}^{n_y}$.

We derive a stability result for this system. As in [1], there exists a smooth, bounded matrix function $R(t) \in \mathcal{R}^{(n_x - n_y) \times n_x}$ whose linearly independent, normalized rows form a basis for the nullspace of B^T (R can be taken to be orthonormal). Thus, for each t, $0 \le t \le t_f$,

$$RB = 0. \tag{2.4}$$

We assume that there exists a constant K_1 of moderate size such that

$$\|(CB)^{-1}\| \le K_1 \tag{2.5}$$

uniformly in t, and determine (see [1, Lemma 2.1]) that there is a constant K_2 of moderate size such that

$$\left\| \begin{pmatrix} R \\ C \end{pmatrix}^{-1} \right\| \le K_2. \tag{2.6}$$

The constant K_2, in addition to K_1, depends also on $\|B\|$, $\|C\|$, and $\|R\|$. Let K_3 be a moderate bound on B, C, R, and their derivatives:

$$\|B^{(j)}\|, \|C^{(j)}\|, \|R^{(j)}\| \le K_3, \qquad j = 0, 1, \ldots, m. \tag{2.7}$$

Define new variables

$$\mathbf{u} = R\mathbf{x}, \qquad 0 \le t \le t_f. \tag{2.8}$$

Then, using (2.3b), the inverse transformation is given by

$$\mathbf{x} = \begin{pmatrix} R \\ C \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{u} \\ -\mathbf{r} \end{pmatrix} \equiv S\mathbf{u} - F\mathbf{r}, \tag{2.9}$$

where $S(t) \in \mathcal{R}^{n_x \times (n_x - n_y)}$ satisfies

$$RS = I, \qquad CS = 0, \tag{2.10}$$

and

$$F := B(CB)^{-1}. \tag{2.11}$$

By our assumptions and (2.6), this mapping is well conditioned. Both S and F are smooth and bounded. The first m derivatives of S and F are bounded by a constant involving K_2 and K_3. Taking m derivatives of (2.8) yields

$$\mathbf{u}^{(m)} = (R\mathbf{x})^{(m)} = \sum_{j=1}^{m} \left[RA_j + \begin{pmatrix} m \\ j-1 \end{pmatrix} R^{(m-j+1)} \right] \mathbf{z}_j + R\mathbf{q}. \tag{2.12}$$

Further, using $m-1$ derivatives of (2.9) we obtain the *essential underlying ODE*

$$\mathbf{u}^{(m)} = \sum_{j=1}^{m} \left[RA_j + \begin{pmatrix} m \\ j-1 \end{pmatrix} R^{(m-j+1)} \right] [(S\mathbf{u})^{(j-1)} - (F\mathbf{r})^{(j-1)}] + R\mathbf{q}. \tag{2.13}$$

For a unique solution of (2.3) one needs to impose $m(n_x - n_y)$ independent boundary conditions

$$B_0\mathbf{z}(0) + B_1\mathbf{z}(t_f) = \beta. \tag{2.14}$$

These could be, for instance, initial conditions which, together with (2.3b) and its first $m-1$ derivatives all sampled at $t = 0$, form mn_x initial conditions which specify $\mathbf{z}(0)$. The boundary conditions can be written as $m(n_x - n_y)$ conditions on $\mathbf{u}$ and its first $m-1$ derivatives needed to specify a unique solution for the EUODE (2.13). If this ODE problem is stable, i.e., if Green's function $\mathcal{G}(t, s)$ and its first $m-1$ derivatives in t are bounded in norm by a constant of moderate size, say K_4 (cf. [3, Chap. 3]), then a similar conclusion holds for the DAE. We obtain the following theorem.

THEOREM 2.1. *Let the* DAE (2.3) *have smooth, bounded coefficients, and assume that* (2.5) *holds and that the underlying problem for* (2.13) *is stable. Then there is a constant K of moderate size such that*

$$\|\mathbf{z}\| \leq K \left(\|\mathbf{q}\| + \sum_{j=0}^{m-1} \|\mathbf{r}^{(j)}\| + |\beta| \right), \tag{2.15a}$$

$$\|\mathbf{y}\| \leq K \left(\|\mathbf{q}\| + \sum_{j=0}^{m} \|\mathbf{r}^{(j)}\| + |\beta| \right). \tag{2.15b}$$

Proof. Our assumptions guarantee the well-conditioning of the transformation from $\mathbf{x}$ to $\mathbf{u}$ and back. The boundary data for $\mathbf{u}$ is therefore bounded by $\sum_{j=0}^{m-1} \|\mathbf{r}^{(j)}\|+$

$|\beta|$ times a moderate constant. We may write $\mathbf{u}(t)$ in terms of Green's function $\mathcal{G}(t,s)$, differentiate $m-1$ times, and take norms, obtaining

$$\|\mathbf{u}^{(l)}\| \le \tilde{K}\left(\|\mathbf{q}\| + \sum_{j=0}^{m-1}\|\mathbf{r}^{(j)}\| + |\beta|\right), \qquad 0 \le l \le m-1$$

with $\tilde{K}$ a moderate constant depending on K_2, K_3, and K_4. Conclusion (2.15a) is then obtained using (2.9).

Now, given $\mathbf{x}$ we obtain $\mathbf{y}$ through multiplying (2.3a) by C, yielding

$$\mathbf{y} = (CB)^{-1}C\left(\mathbf{x}^{(m)} - \sum_{j=1}^{m} A_j\mathbf{z}_j - \mathbf{q}\right). \tag{2.16}$$

Differentiating (2.3b) m times we substitute for $C\mathbf{x}^{(m)}$ in (2.16), and using (2.15a), obtain the bound (2.15b). □

Remark. The EUODE (2.13) is not unique. For any nonsingular, smooth, well-conditioned transformation $T(t) \in \mathcal{R}^{(n_x-n_y)\times(n_x-n_y)}$, the transformed $R(t)$ given by

$$R \leftarrow TR \tag{2.17}$$

still satisfies (2.4), (2.6), and (2.7). Hence R is unique only up to such a transformation and, correspondingly, so is the EUODE. However, a transformation of the variables $\mathbf{u}$ in (2.8) corresponding to (2.17) does not alter the existence (or lack thereof) of a bound of moderate size on the Green's function, and hence the stability properties are properly reflected in Theorem 2.1. For later theoretical purposes, we may wish to choose T such that the EUODE (2.13) is amenable to a direct discretization. In particular, for $m = 1$ and a BDF discretization we can choose T so that the resulting matrix $(RA_1 + R')S$ is essentially diagonally dominant or block upper triangular (see [14], [15], and [3]). □

We remark that a bound similar to (2.15) may also be obtained using Theorem 2.1 of [1] applied to the index-2 DAE

$$\mathbf{z}_j' = \mathbf{z}_{j+1} + B\mu_j, \qquad j = 1, \ldots, m-1, \tag{2.18a}$$

$$\mathbf{z}_m' = \sum_{j=1}^{m} A_j\mathbf{z}_j + B\mathbf{y} + \mathbf{q}, \tag{2.18b}$$

$$\mathbf{0} = C\mathbf{z}_1 + \mathbf{r}, \tag{2.18c}$$

$$\mathbf{0} = C\mathbf{z}_2 + C'\mathbf{z}_1 + \mathbf{r}', \tag{2.18d}$$

$$\vdots$$

$$\mathbf{0} = \sum_{j=1}^{m}\binom{m-1}{j-1} C^{(m-j)}\mathbf{z}_j + \mathbf{r}^{(m-1)} \tag{2.18e}$$

subject to the original boundary conditions (2.14). Here we have applied a particular so-called *stabilized index reduction* technique [11], adding multiplier functions $\mu_j(t) \in \mathcal{R}^{n_y}$ to compensate for insisting that the constraint (2.3b) and its first $m-1$ derivatives be all satisfied at all t. This DAE problem has the same exact solution as the original higher-index problem (2.3), (2.14) because differentiation and substitution for each of the algebraic constraints in (2.18) yields $CB\mu_j = 0$, which implies $\mu_j = 0$, $j = 1, \dots, m-1$. The EUODE for (2.18) is obtained using R of (2.4) m times, i.e., for the variables

$$\mathbf{w}_j := R\mathbf{z}_j = R\mathbf{x}^{(j-1)}, \qquad j = 1, \dots, m, \tag{2.19}$$

we obtain

$$\mathbf{w}'_j = R\mathbf{z}_{j+1} + R'\mathbf{z}_j, \qquad j = 1, \dots, m-1, \tag{2.20a}$$

$$\mathbf{w}'_m = \sum_{j=1}^{m} RA_j\mathbf{z}_j + R'\mathbf{z}_m + R\mathbf{q}, \tag{2.20b}$$

where $\mathbf{z}_j$ are expressed in terms of $\mathbf{w}$ using the recursive relation

$$\mathbf{z}_j = S\mathbf{w}_j - F\left[\mathbf{r}^{(j-1)} + \sum_{l=1}^{j-1} \begin{pmatrix} j-1 \\ l-1 \end{pmatrix} C^{(j-l)}\mathbf{z}_l\right], \qquad j = 1, \dots, m. \tag{2.21}$$

It is easily shown that the "stabilized" index-2 form (2.18) with (2.14) is stable whenever the original high-index equation (2.3) is.

3. Other transformations. The EUODE (2.13) uses a minimal number of constraint differentiations. Therefore, we view the assumption that it is stable with the given boundary operator as essential. From this we now derive stability for a number of other problem reformulations that have appeared in the literature.

3.1. Baumgarte stabilization. The most straightforward transformation of the DAE (2.3) into an ODE involves replacing the constraint

$$\mathbf{g}(\mathbf{x}, t) \equiv C\mathbf{x} + \mathbf{r} = \mathbf{0}$$

with its mth time derivative plus initial conditions:

$$\mathbf{g}^{(m)} = \frac{d^m\mathbf{g}(\mathbf{x}(t), t)}{dt^m} = \mathbf{0}, \tag{3.1a}$$

$$\mathbf{g}(\mathbf{x}(0), 0) = \frac{d}{dt}\mathbf{g}(\mathbf{x}(0), 0) = \cdots = \frac{d^{m-1}}{dt^{m-1}}\mathbf{g}(\mathbf{x}(0), 0) = \mathbf{0}. \tag{3.1b}$$

However, this causes well-known drift difficulties. A generalization of Baumgarte's method [4] replaces (3.1a) with the equation

$$\sum_{j=0}^{m} \alpha_j \frac{d^j}{dt^j}\mathbf{g}(\mathbf{x}(t), t) = \mathbf{0}, \tag{3.2}$$

where α_j are chosen so that $\alpha_m = 1$ and the roots of the polynomial

$$\sigma(\tau) = \sum_{j=0}^{m} \alpha_j \tau^j \tag{3.3}$$

are all nonpositive. For instance, one may choose $\sigma(\tau) = (\tau + \gamma)^m$ for some $\gamma \geq 0$. We now investigate the stability of (2.3a), (3.2), (2.14), and (3.1b).

In (3.2) we have an expression for $C\mathbf{x}^{(m)}$ that we may substitute into (2.3a) multiplied by C and eliminate $\mathbf{y}$:

$$\begin{aligned} \mathbf{y} = -(CB)^{-1}\Bigg\{ & \textstyle\sum_{j=1}^{m} \left[CA_j + \binom{m}{j-1} C^{(m-j+1)} \right] \mathbf{z}_j \\ & + C\mathbf{q} + \mathbf{r}^{(m)} + \textstyle\sum_{l=0}^{m-1} \alpha_l \mathbf{g}^{(l)} \Bigg\}. \end{aligned} \tag{3.4}$$

Substituting back into (2.3a) we obtain an ODE for $\mathbf{x}$

$$\begin{aligned} \mathbf{x}^{(m)} &= \sum_{j=1}^{m} \left[HA_j - \binom{m}{j-1} FC^{(m-j+1)} \right] \mathbf{z}_j \\ &\qquad + H\mathbf{q} - F\mathbf{r}^{(m)} - F \sum_{l=0}^{m-1} \alpha_l \mathbf{g}^{(l)}, \end{aligned} \tag{3.5a}$$

$$\mathbf{g}^{(l)} = \sum_{j=1}^{l+1} \binom{l}{j-1} C^{(l-j+1)} \mathbf{z}_j + \mathbf{r}^{(l)}, \tag{3.5b}$$

with F given by (2.11) and H the projection

$$H = I - FC = SR. \tag{3.6}$$

We then ask the question regarding the stability of the ODE problem (3.5), (2.14), (3.1b). Obviously, the differentiation in (3.2) has enlarged the size of the ODE problem, so the question really is whether the new solution modes thus introduced (together with (3.1b)) cause an instability.

To resolve this question, define

$$\mathbf{u} = R\mathbf{x}, \qquad \mathbf{v} = C\mathbf{x}. \tag{3.7}$$

Then

$$\mathbf{x} = S\mathbf{u} + F\mathbf{v} \tag{3.8}$$

(cf. (2.9)). So, by (2.6),

$$\|\mathbf{x}\| \le K_2(\|\mathbf{u}\| + \|\mathbf{v}\|).$$

To see what $\mathbf{u}$ and $\mathbf{v}$ satisfy, multiply (3.5a) by R and by C. This gives

$$\mathbf{u}^{(m)} = \sum_{j=1}^{m} \left[RA_j + \binom{m}{j-1} R^{(m-j+1)} \right] [(S\mathbf{u})^{(j-1)} + (F\mathbf{v})^{(j-1)}] + R\mathbf{q}, \tag{3.9a}$$

$$\mathbf{v}^{(m)} = -\sum_{j=0}^{m-1} \alpha_j \mathbf{v}^{(j)} - \sum_{j=0}^{m} \alpha_j \mathbf{r}^{(j)}, \tag{3.9b}$$

$$\mathbf{v}^{(j)}(0) = -\mathbf{r}^{(j)}(0), \qquad j = 0, \dots, m-1. \tag{3.9c}$$

Now, letting $t_f \to \infty$, we have in (3.9b), (3.9c) a uniformly stable initial value problem for $\mathbf{v}$ as long as at most one root of $\sigma(\tau)$ is 0 and the rest have negative real parts. For instance, with

$$\sigma(\tau) = (\tau + \gamma)^m, \tag{3.10}$$

any choice of $\gamma > 0$ yields a uniformly, *asymptotically* stable problem for $\mathbf{v}$, while the choice $\gamma = 0$, which corresponds to using (3.1a) in place of (3.2), allows for a mild instability, viz. a polynomial error growth (of degree $m-1$), to occur. Moreover, if the EUODE is asymptotically stable and $\gamma > 0$, then the ODE (3.5) is also asymptotically stable. In other words, the solution modes introduced by the Baumgarte technique are strictly decreasing if $\gamma > 0$.

Once $\mathbf{v}$ is integrated it may be substituted into (3.9a) to obtain the EUODE for $\mathbf{u}$. The problem for (3.9) is therefore stable, and by (3.8), so is the problem that the Baumgarte technique yields.

This analysis agrees with practical observations. First, the direct index reduction (3.1) has only a mild instability for a stable original problem. This instability gets worse as m increases, i.e., it is worse for the DAE with position constraints (1.1a), (1.1b) than for the DAE with motion constraints (1.1a), (1.1c). Second, any $\gamma > 0$ in (3.10) yields a stable problem in (3.5). The difference in performance for different values of $\gamma > 0$, when there is any, is due to discretization effects applied to stable ODE problems. This is taken up in §4, but we may already expect here that if K in (2.15) is indeed of moderate size and the discretization mesh is very fine, then the results will not be sensitive to the choice of γ. In practice, the choice of γ in sensitive situations is far from clear.

3.2. Reduction to index 2. In (3.1) and (3.2) we have differentiated $\mathbf{g}(\mathbf{x}, t)$ m times, reducing the index to 1. A subsequent elimination of $\mathbf{y}$ gives an ODE. If instead we differentiate the constraints only $m-1$ times, we obtain a DAE of index 2 consisting of (2.3a) and

$$\sum_{j=0}^{m-1} \hat{\alpha}_j \frac{d^j}{dt^j} \mathbf{g}(\mathbf{x}(t), t) = \mathbf{0}, \tag{3.11}$$

with $\hat{\alpha}_{m-1} = 1$. This is subject to (2.14) and

$$\mathbf{g}(\mathbf{x}(0), 0) = \cdots = \frac{d^{m-2}}{dt^{m-2}} \mathbf{g}(\mathbf{x}(0), 0) = \mathbf{0}. \tag{3.12}$$

The stability analysis for this problem formulation proceeds precisely as before: using the transformation (3.7), (3.8) we obtain (3.9a) and

$$\sum_{j=0}^{m-1} \hat{\alpha}_j \mathbf{v}^{(j)} = -\sum_{j=0}^{m-1} \hat{\alpha}_j \mathbf{r}^{(j)}, \tag{3.13a}$$

$$\mathbf{v}^{(j)}(0) = -\mathbf{r}^{(j)}(0), \qquad j = 0, \ldots, m-2. \tag{3.13b}$$

The ODE (3.13a) is asymptotically stable if the roots of $\hat{\sigma}(\tau) = \sum_{j=0}^{m-1} \hat{\alpha}_j \tau^j$ all have negative real parts. Considering in particular

$$\hat{\sigma}(\tau) = (\tau + \gamma)^{m-1}, \tag{3.14}$$

there is asymptotic stability if $\gamma > 0$ and a polynomial growth of order $m-2$ if $\gamma = 0$. The latter corresponds to direct index reduction. In particular, for mechanical systems with $m = 2$ one direct differentiation of the constraints ($\gamma = \hat{\alpha}_0 = 0$) yields a stable, although not asymptotically stable, problem (3.13) for $\mathbf{v}$.

The stability of the index-2 problem (2.3a), (3.11), (2.14), (3.12) follows, as before, from that of (3.13) and the analysis of §2.

The justification for considering this type of index reduction is that certain implicit discretization schemes like BDF may already be successfully applied to the resulting formulation (cf. [6], [16], and [1]). This is considered in §4.

Another problem reformulation that reduces the index to 2 is, of course, the stabilized index reduction technique of (2.18). In §3.5 below we consider an entire family of additional stabilized index reductions.

3.3. State-space form. The problem formulations considered hitherto in this section all end up in an ODE of size mn_x, requiring supplementary boundary conditions. In contrast, the EUODE (2.13) only has size $m(n_x - n_y)$, and no supplementary conditions are required for the problem reformulation. Moreover, incorporation of the constraint (2.3b) and its first $m-1$ derivatives into the transformation has insured no drift in a subsequent discretization.

This can be done more generally: Let $\tilde{R}(t)$ be a smooth, bounded function, $\tilde{R}(t) \in \mathcal{R}^{(n_x-n_y)\times n_x}$, such that

$$\left\| \begin{pmatrix} \tilde{R} \\ C \end{pmatrix}^{-1} \right\| \leq \tilde{K}, \quad \|\tilde{R}^{(j)}\| \leq \tilde{K}, \quad j = 0,1,\ldots,m \tag{3.15}$$

for a constant $\tilde{K}$ of moderate size. (We do not require $\tilde{R}B = 0$.) Define

$$\tilde{\mathbf{u}} = \tilde{R}\mathbf{x}, \qquad \mathbf{x} = \begin{pmatrix} \tilde{R} \\ C \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\mathbf{u}} \\ -\mathbf{r} \end{pmatrix} \equiv \tilde{S}\tilde{\mathbf{u}} - \tilde{F}\mathbf{r}. \tag{3.16}$$

Taking $\alpha_0 = \cdots = \alpha_{m-1} = 0$ in (3.5a) (i.e., $\gamma = 0$ in (3.10)), we multiply it by $\tilde{R}$ to obtain

$$\tilde{\mathbf{u}}^{(m)} = \sum_{j=1}^{m} \left[\tilde{R}HA_j + \begin{pmatrix} m \\ j-1 \end{pmatrix} (\tilde{R}^{(m-j+1)} - \tilde{R}FC^{(m-j+1)}) \right] \mathbf{z}_j$$

$$+\tilde{R}H\mathbf{q} - \tilde{R}F\mathbf{r}^{(m)} \tag{3.17a}$$

$$\mathbf{z}_j = (\tilde{S}\tilde{\mathbf{u}})^{(j-1)} - (\tilde{F}\mathbf{r})^{(j-1)}, \qquad 1 \leq j \leq m. \tag{3.17b}$$

This state-space ODE is subject to the boundary conditions (2.14), suitably transformed. The EUODE is obtained as a special case with $\tilde{R} = R$.

The stability of the problem formulation (3.17) follows immediately upon relating $\tilde{\mathbf{u}}$ and $\mathbf{u}$ through $\mathbf{x}$, i.e., using (3.16), (2.8), (2.9), and their derivatives. The obtained stability bound depends on $\tilde{K}$ (and of course on K of (2.15)).

A favorite practical choice for $\tilde{R}$ is as a piecewise constant function [20], [18]. Thus, choosing $\tilde{R}$ at a certain reference time t_c so that (3.15) is satisfied, one proceeds to integrate in t, holding this $\tilde{R}$ constant so long as (3.15) holds with a reasonable $\tilde{K}$. When (3.15) is deemed violated, a new constant matrix $\tilde{R}$ is chosen based on a new reference point, giving a different ODE (3.17). The segments are connected in such switching points through continuity of $\mathbf{z}$. The lack of nonzero derivatives of $\tilde{R}$ over the integrated segment gives (3.17) an attractive form. A robust detection scheme for the necessity to change $\tilde{R}$ may prove to be the more difficult aspect of such a procedure, as discussed in §4.

3.4. Overdetermined DAE. Consider deriving the EUODE from the first-order form

$$\mathbf{z}'_j = \mathbf{z}_{j+1}, \qquad j = 1, \ldots, m-1, \tag{3.18a}$$

$$\mathbf{z}'_m = \sum_{j=1}^{m} A_j \mathbf{z}_j + B\mathbf{y} + \mathbf{q}. \tag{3.18b}$$

We proceed to define $\mathbf{w}_j = R\mathbf{z}_j$ and obtain the back-transformation using

$$\mathbf{0} = \sum_{l=1}^{j} \begin{pmatrix} j-1 \\ l-1 \end{pmatrix} C^{(j-l)} \mathbf{z}_l + \mathbf{r}^{(j-1)}, \qquad j = 1, \ldots, m. \tag{3.19}$$

The transformation matrix for each j is $\binom{R}{C}$. If we now write down the equations to be satisfied, (3.18), (3.19), they form an overdetermined DAE (ODAE). This overdetermination is subsequently resolved when multiplying (3.18a) and (3.18b) by R, obtaining the EUODE (2.20) in terms of

$$\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_m)^T. \tag{3.20}$$

The fact that (3.18), (3.19) is indeed an ODAE is reflected by the fact that we could replace (3.18a) by the "stabilized form"

$$\mathbf{z}'_j = \mathbf{z}_{j+1} + B\mu_j, \tag{3.21}$$

and obtain precisely the same EUODE, as in (2.18). Note that the DAE (3.21), (3.18b), (3.19) is not overdetermined any more, and that we have not used the fact that $\mu_j = \mathbf{0}$.

The ODAE (3.18), (3.19) subject to (2.14) has a unique solution, but when we replace it by a discretized form using one of the conventional difference schemes, we cannot expect an exact solution to exist. Still, one can multiply the discretized (3.18) by R, and (3.21) can replace (3.18a) provided that R and B are sampled at the same point t. The discretized DAE (3.21), (3.18b), (3.19) is therefore equivalent to a particular projection for solving the problem of minimizing the residual of the discretized ODAE subject to satisfying the discretized (3.19) (cf. [9] and [10]).

Similar arguments apply when replacing R by a more general smooth, bounded $\tilde{R}$ satisfying (3.15). Before applying $\tilde{R}$ we must differentiate the constraints once more and substitute (3.1) into (3.18b) to eliminate $\mathbf{y}$, obtaining

$$\mathbf{z}'_m = \sum_{j=1}^{m} \left[HA_j - \begin{pmatrix} m \\ j-1 \end{pmatrix} FC^{(m-j+1)} \right] \mathbf{z}_j + H\mathbf{q} - F\mathbf{r}^{(m)}. \tag{3.22}$$

A particular projected solution for the discretized ODAE (3.18a), (3.19), (3.22) with (3.19) treated as constraints is then obtained from the same discretization applied to (3.17) written in first-order form, and this in turn is equivalent to the discretized form of the DAE (3.22), (3.19) and

$$\mathbf{z}'_j = \mathbf{z}_{j+1} + \tilde{B}\mu_j, \qquad j = 1, \ldots, m-1, \tag{3.23}$$

with $\tilde{B}(t) \in \mathcal{R}^{n_x \times n_y}$ having full rank and satisfying

$$\tilde{R}\tilde{B} = 0. \tag{3.24}$$

For other possibilities, see [9], and [10]. Here we note that the stability treatment of the ODAEs we have described is covered by our previous stability analysis.

3.5. Projected invariants. Consider the following general procedure: Differentiating the constraints (2.3b) m times and eliminating $\mathbf{y}$ from (3.18b), we obtain (3.22), which, together with (3.18a), forms an explicit ODE system for $\mathbf{z}$. However, this allows for unacceptable drifts in the constraints after discretization, so we reimpose the first k constraints of (3.19), for some integer $k \leq m$,

$$\mathbf{0} = \sum_{l=1}^{j} \begin{pmatrix} j-1 \\ l-1 \end{pmatrix} C^{(j-l)}\mathbf{z}_l + \mathbf{r}^{(j-1)}, \qquad j = 1, \ldots, k. \tag{3.25}$$

The constraints (3.25) form an invariant of the ODE (3.18a), (3.22) (cf. Gear [12]). To satisfy these constraints even after discretization, we project the ODE as follows.

For k given smooth, full-rank bounded matrix functions $R_j(t) \in \mathcal{R}^{(n_x-n_y)\times n_x}$ satisfying (3.15) (for $\tilde{R} = R_j$), require that

$$\text{(3.26a)} \quad R_j\mathbf{z}_j' = R_j\mathbf{z}_{j+1}, \qquad j = 1, \ldots, \min(k, m-1),$$

$$\text{(3.26b)} \quad R_m\mathbf{z}_m' = R_m \left\{ \sum_{j=1}^{m} \left[HA_j - \begin{pmatrix} m \\ j-1 \end{pmatrix} FC^{(m-j+1)} \right] \mathbf{z}_j + H\mathbf{q} - F\mathbf{r}^{(m)} \right\} \quad (\text{if } k = m).$$

This is equivalent to writing

$$\text{(3.27a)} \quad \mathbf{z}_j' = \mathbf{z}_{j+1} + B_j\mu_j, \qquad j = 1, \ldots, m-1,$$

$$\text{(3.27b)} \quad \mathbf{z}_m' = \sum_{j=1}^{m} \left[HA_j - \begin{pmatrix} m \\ j-1 \end{pmatrix} FC^{(m-j+1)} \right] \mathbf{z}_j + H\mathbf{q} - F\mathbf{r}^{(m)} + B_m\mu_m,$$

where $B_j = 0$ if $j > k$, but for $1 \leq j \leq k$, $B_j(t) \in \mathcal{R}^{n_x\times(n_x-n_y)}$ have full rank and satisfy for each t

$$B_j = N_jC^T, \qquad R_jB_j = 0, \tag{3.28}$$

with N_j smooth well-conditioned matrices. The additional unknowns $\mu_j(t) \in \mathcal{R}^{n_y}$ are multipliers.

Using a discretization on a mesh, the discretized equations (3.26) or (3.27) are required to hold together with (3.25) at all mesh points.

Clearly, the obtained system (3.27), (3.25) is a DAE of index 2 in Hessenberg form, which, together with (2.14), is well conditioned if the original problem is. Also, the projected invariant approach can be viewed as an ODAE approach, although we feel that it gives more insight. The advantage here compared with the stabilized index reduction (2.18) is that there the stabilizer B is dictated by the problem while here we may choose B_j (i.e., R_j, as long as (3.15) is satisfied). This proves useful in cases where C^T behaves very differently from B, because here we may in fact choose $B_j = C^T$ (see Examples 3 and 4 in §4).

Summarizing the results of this section, we have seen that the stability of the original problem is preserved by problem reformulations such as stabilized index reduction, introduction of (properly chosen) Baumgarte parameters, and reduction to state-space form. Direct index reduction leads to a mild instability (i.e., the possible error growth is polynomial, not exponential)[2], which becomes progressively worse for higher-index problems. Finally, overdetermined DAEs can be regarded as a special case of one of the above forms, depending on the projection that is used in the numerical solution procedure. In the next section, we will consider the stability of discretization methods applied to these various formulations.

4. Discretization.

4.1. Backward Euler for an index-2 DAE.

To better understand the stability behavior of numerical methods applied to the above formulations, consider the Hessenberg index-2 system

$$\mathbf{x}' = A\mathbf{x} + B\mathbf{y} + \mathbf{q}, \tag{4.1a}$$

$$\mathbf{0} = C\mathbf{x} + \mathbf{r}, \tag{4.1b}$$

which is a special case of (2.3) for $m = 1$ and may arise from stabilized or direct reduction of a higher-index system to index 2. For simplicity of presentation, we will consider discretizing (4.1) by the backward Euler method, which gives

$$\mathbf{x}_n = \mathbf{x}_{n-1} + hA_n\mathbf{x}_n + hB_n\mathbf{y}_n + h\mathbf{q}_n, \tag{4.2a}$$

$$\mathbf{0} = C_n\mathbf{x}_n + \mathbf{r}_n. \tag{4.2b}$$

Note that, if we first derive an explicit ODE in $\mathbf{x}$ by differentiating (4.1b), use this to eliminate $\mathbf{y}$, and then discretize using backward Euler, we get

$$\mathbf{x}_n = (I - hHA + hFC')^{-1}[\mathbf{x}_{n-1} + hH\mathbf{q} - hF\mathbf{r}']$$

(all quantities are sampled at t_n, unless otherwise noted). So the amplification matrix is

$$(I - hHA + hFC')^{-1}. \tag{4.3}$$

But for (4.2) we obtain, upon multiplying (4.2a) by C and substituting (4.2b) to eliminate $\mathbf{y}_n$,

$$\mathbf{x}_n = H(\mathbf{x}_{n-1} + hA\mathbf{x}_n + h\mathbf{q}) - F\mathbf{r}.$$

Then, using

$$H_n\mathbf{x}_{n-1} = \mathbf{x}_{n-1} - F(C_{n-1} + hC'_{n-1} + O(h^2))\mathbf{x}_{n-1}$$

yields

$$\mathbf{x}_n = (I - hHA)^{-1}[(I - hFC' + O(h^2))\mathbf{x}_{n-1} + hH\mathbf{q} - F(\mathbf{r}_n - \mathbf{r}_{n-1})],$$

so the amplification matrix is approximately

$$(I - hHA)^{-1}(I - hFC'). \tag{4.4}$$

[2]We note that this result cannot be achieved by merely looking at local eigenvalues. For an example, see [10]. We also note that in some engineering applications, any significant drift from the original constraint manifold may be considered unacceptable; thus even the mild instability may pose a problem.

Taking $A = 0$ for simplicity, we see that, while in (4.3) we have the backward Euler matrix for the ODE

$$\mathbf{x}' = -FC'\mathbf{x}, \tag{4.5}$$

in (4.4) we have the forward Euler matrix for the same ODE. If (4.5) is stiff then the backward Euler scheme for (4.1) behaves like a nonstiff method!

The same phenomenon can also be seen as follows. Let $\mathbf{u}_n = R_n\mathbf{x}_n$, where $R_n = R(t_n)$ (cf. (2.4), (2.8)). Then $\mathbf{x}_n = S_n\mathbf{u}_n - F_n\mathbf{r}_n$. Multiplying (4.2a) by R_n and changing variables to $\mathbf{u}$, we find that

$$\mathbf{u}_n = \mathbf{u}_{n-1} + h((R'S)_{n-1} + O(h))\mathbf{u}_{n-1} + hRAS\mathbf{u}_n - h(R' + O(h))F\mathbf{r} + hR\mathbf{q}. \tag{4.6}$$

We note that (4.6) is a consistent discretization of the EUODE, but it is not the same as backward Euler applied directly to the EUODE because in (4.6), the term involving $R'S$ is discretized *explicitly*. Thus for problems where $R'S$ is large but the solution is smooth, we would expect that the step size for (4.2) must be restricted to maintain numerical stability.[3] A similar problem of numerical instability arises when the higher-index problem is discretized directly by such a method.

The analysis using $\mathbf{u}$ has the advantage that the amplification matrix has a smaller size (because there are fewer unknowns). But it depends on the choice of R as per (2.17), whereas (4.3)–(4.5) are independent of the choice of R. When considering (4.6) and $R'S$ one must avoid premature conclusions about suitability of the backward Euler scheme, although a positive stability conclusion (upon finding that $R'S$ is not large for a given problem) is immediate.

It is natural to ask under what conditions and for which formulations FC' (or $R'S$ for the best scaling) can become large. The question is more immediately answered for $FC' = B(CB)^{-1}C'$. If we assume that the solution $\mathbf{x}$ varies at a rate similar to that of C, so that the step size taken for accuracy reasons satisfies $h||C'|| << 1$, then FC' can be large only if $||B||$ or $||(CB)^{-1}||$ are large (and forming the product $B(CB)^{-1}$ does not cancel this effect). Assume also that $||C||, ||(CC^T)^{-1}|| = O(1)$. In such a case the projected invariant approach (3.27) with $k = m = 1$ and $B_1 = C^T$ is rather useful: the obtained index-2 DAE, which is subsequently discretized, is

$$\mathbf{x}' = (HA - FC')\mathbf{x} + H\mathbf{q} - F\mathbf{r}' + C^T\mu, \tag{4.7a}$$

$$\mathbf{0} = C\mathbf{x} + \mathbf{r}, \tag{4.7b}$$

so C^T plays the role that B plays in (4.1) and a BDF discretization is expected to behave like a stiff solver because $C^T(CC^T)^{-1}C'$ is not large in norm. This is demonstrated below, in Example 2. The price paid to obtain (4.7) does include an additional differentiation of the constraints.

We note further that for systems such as (4.7), where $B = C^T$, $R'S$ is of moderate size, *even if* $||(CC^T)^{-1}||$ *is large*. We show this for the case where C has a smooth singular value decomposition (SVD)

$$C^T = U\begin{pmatrix}\Sigma\\0\end{pmatrix}V^T$$

[3]This property of inherently explicit treatment of $R'S$ when the index-2 problem is discretized directly is shared also by higher-order BDF and by most implicit Runge–Kutta schemes. We note also that because of the strong relationship between semi-explicit index-2 problems and fully implicit index-1 problems [11], this problem of numerical instability can also be expected to occur for certain fully implicit index-1 DAEs.

(which is guaranteed if C is analytic [7]). We may choose $R = (\,0 \quad I\,)U^T$. Then (for each t)

$$(\,F \quad S\,) = \begin{pmatrix} C \\ R \end{pmatrix}^{-1} = U \begin{pmatrix} (V\Sigma)^{-1} & 0 \\ 0 & I \end{pmatrix},$$

so

$$S = U \begin{pmatrix} 0 \\ I \end{pmatrix}.$$

It follows that

$$R'S = (\,0 \quad I\,)U'^TU \begin{pmatrix} 0 \\ I \end{pmatrix}.$$

Hence for most reasonable C, the performance of numerical methods based on discretization of the projected invariants formulations should not degrade due to stability when approaching a rank-deficiency in C (i.e., no smaller step sizes are enforced due to stability).

4.2. Discretization and stiffness. A number of numerical methods currently implemented in CAD codes consist of more or less standard stiff or nonstiff discretizations applied to one of the formulations in §3. By a "nonstiff discretization" we mean a difference scheme (e.g., explicit Runge–Kutta) that works efficiently for a nonstiff initial value ODE, but becomes inefficient for a stiff ODE because absolute stability restrictions force a step size selection h that is much smaller than what accuracy requirements alone would dictate. A "stiff discretization," e.g., a BDF scheme, does not usually suffer from such absolute stability restrictions and is inherently an implicit difference scheme.

We now consider such methods:

1. Baumgarte stabilization, followed by (i) a nonstiff discretization or by (ii) a stiff discretization.

2. Safe reduction to index 2 (as in §§3.5 or 3.2 or in (2.18)), followed by a stiff discretization.

3. Reduction to state space form, followed by (i) a nonstiff discretization or by (ii) a stiff discretization.

Given that we consider essentially the same discretization schemes applied to problem reformulations that we have just proved equivalent under mild conditions, one might expect all of these methods to perform equally well. As it turns out, however, it is surprisingly easy to give examples (as we shall do below) where each of the three methods significantly outperforms the other two. Indeed, it is often not very clear in the literature what is meant by the term "stiffness" when it is applied to a higher-index DAE. To understand this, we distinguish among four cases for the EUODE (2.13) (or (2.20)).

1. The EUODE is nonstiff; B and C vary slowly.

2. The EUODE is nonstiff; B or C do not vary slowly.

3. The "frozen coefficients part" of the EUODE, viz. $\hat{\mathbf{u}}^{(m)} = \sum_{j=1}^{m} RA_jS\hat{\mathbf{u}}^{(j-1)}$ (or the homogeneous (2.20) with constant (frozen) coefficients) is stiff; B and C vary slowly. In this case, the stiffness is caused by the "ODE part" of the system.

4. The "variable coefficients part" of the EUODE, i.e., what remains after subtracting out the frozen coefficient part, is stiff. In this case, the stiffness is caused by time- or solution-dependent coupling of the constraints with the differential equations.

Case 1. Many mechanical systems yield ODEs that are not stiff. If no part of the mechanical system moves rapidly in time and the system is not heterogeneous, we may expect a nonstiff ODE to result in all problem formulations of §3. In this case, a Baumgarte stabilization (3.5) with, say, $\gamma = 1$ in (3.10), can be efficiently solved using a nonstiff discretization. For such examples, see [4]. Note that γ should not be taken large in this case, because this may introduce artificial stiffness (cf. (3.9b)).

While the Baumgarte technique yields a nonstiff ODE (so, for instance, an explicit difference scheme may be applied to (3.5)), the other two reformulations require satisfaction of constraints and therefore have an implicit part, even if the reduced ODE is discretized using an explicit scheme [18] (similarly [16]). In simple situations (where the mth constraint differentiation is not a bother either), such a Baumgarte technique is therefore more efficient.

Case 2. Generally, a robust discretization would have to use a step size commensurate with the variation of B and C. With such a step size, a Baumgarte technique or an index-2 reduction method should perform well, as above, except that the additional constraint differentiation or a poorly scaled choice of the parameter in the Baumgarte technique might increase errors.

With C varying significantly, however (e.g., corresponding to a rapidly rotating shaft), the robustness of a state-space form reduction using a constant $\tilde{R}$ may be called into question.

Example 1. For

$$C(t) = (\sin(\nu t), \cos(\nu t)), \qquad 0 \le t,$$

with $\nu \ge 1$ a parameter, an appropriate choice for $\tilde{R}$ satisfying (3.15) at $t_c = 0$ is

$$\tilde{R} = (1, 0), \qquad 0 \le t \le 1.$$

But then,

$$\begin{pmatrix} \tilde{R} \\ C \end{pmatrix}$$

is singular at $t = (j + 1/2)\pi/\nu$ for all j integers. It is clear that one must restart $\tilde{R}$ (i.e., switch coordinates) at steps $O(1/\nu)$ apart. While the discretization step for any of the other methods must be $O(1/\nu)$ as well, a simple discretization step involves much less effort than a restart.

What is potentially worse, detecting restart points is not easy in practice. (This is somewhat similar to using a Riccati method for stiff boundary value ODEs; see [8].) To see what happens when a singularity point is missed, we continue the example as follows:

$$\mathbf{x}' = -\mathbf{x} + By + \mathbf{q},$$

$$0 = C\mathbf{x} + r,$$

with $B = C^T$, $x_1(0) = 1$, and $\mathbf{q}$ and r are chosen to be

$$\mathbf{q} = \begin{pmatrix} 2e^t + \frac{\sin(\nu t)e^t}{(2-t)} \\ 2e^t + \frac{\cos(\nu t)e^t}{(2-t)} \end{pmatrix},$$

$$r = -(\sin(\nu t) + \cos(\nu t))e^t$$

such that the solution is $\mathbf{x}^T = e^t(1,1)$, $y = -e^t/(2-t)$. With

$$R(t) = (\cos(\nu t), -\sin(\nu t)),$$

we have $S^T = R$, $F^T = C$, and the homogeneous part of the EUODE is

$$u' = -u,$$

with $u(0)$ given. This problem is stable, with $K = O(\nu)$ in (2.15), independently of ν.

The state-space form with $\tilde{R} = (1,0)$ gives, on the other hand, an ODE whose homogeneous part is

$$\tilde{u}' = -(1 + \nu \tan(\nu t))\tilde{u}.$$

So, if one ignores or misses a singularity point, then one may end up integrating an unstable ODE.

In Table 4.1 we list some results obtained using a backward Euler discretization with step size $h = .01$ for $\nu = 1000$. The problem is solved over the interval $t \in [0,1]$. We denote by γ the Baumgarte parameter (i.e., we have replaced the constraint $g(\mathbf{x},t) = 0$ with $g' + \gamma g = 0$ except for the case $\gamma = \infty$, which corresponds to a direct discretization of the given problem). The discretization of (4.7) is referred to as projected invariant. The reported errors are the max-norm of errors in both components of $\mathbf{x}$, and the reported drift is the magnitude of the residual of the original constraint, at the endpoint of the time interval.

TABLE 4.1
Behavior of methods for example 1.

Method	γ	Error	Drift
Baumgarte	0.	.26e+79	.33e+79
Baumgarte	1.	.10e+79	.13e+79
Baumgarte	10.	.37e+75	.48e+75
Baumgarte	100.	.63e+53	.85e+53
Baumgarte	1000.	.27e+08	.13e+09
Baumgarte	10000.	.23e−3	.23e−4
Baumgarte	∞	.20e−3	.14e−15
Projected invariant	NA	.20e−3	0
State-space form ($\tilde{R} = (1,0)$)	NA	.42e+1	0

This example shows the index-2 reduction method in a particularly favorable light: since $|g'| >> |g|$, a rather large γ is needed for the Baumgarte technique to work well. Insisting on satisfying (4.1b) or (4.7b) in the context of an index-2 Hessenberg DAE is advantageous. □

Case 3. We apply the same BDF scheme to discretize the three formulations. It is well known that BDF schemes usually perform well for stiff initial value ODEs. It is less well known that the theory justifying this performance is at present incomplete, and applies mainly to scalar equations. Consider a stiff initial value ODE

$$\mathbf{x}' = A(t)\mathbf{x} \tag{4.8}$$

and its backward Euler discretization

$$h_n^{-1}(\mathbf{x}_n - \mathbf{x}_{n-1}) = A_n \mathbf{x}_n, \tag{4.9}$$

where $0 = t_0 < t_1 < \cdots < t_N = 1$, $h_n = t_n - t_{n-1}$, $A_n := A(t_n)$. Given a nonsingular transformation $T(t)$, let

$$\mathbf{w} = T^{-1}\mathbf{x}. \tag{4.10}$$

Then $\mathbf{w}$ satisfies the ODE

$$\mathbf{w}' = (T^{-1}AT - T^{-1}T')\mathbf{w} \equiv U\mathbf{w}. \tag{4.11}$$

If U is upper triangular with off-diagonal elements that are not too large, or if U is essentially diagonally dominant (see [14], [15], or [3, Chap. 10]),then a backward Euler scheme applied to (4.11), i.e., the discretization is applied *after* the transformation, performs well as a stiff discretization scheme. (To see this we may consider the diagonal part of U first, obtaining stability results for a scalar equation for each of the equations in (4.11), and follow this by a contraction argument for the full U.) But if we apply the transformation (4.10) after the discretization (4.9), we obtain $(\mathbf{w}_n := T_n^{-1}\mathbf{x}_n)$,

$$h_n^{-1}(\mathbf{w}_n - \mathbf{w}_{n-1}) = T_n^{-1}A_nT_n\mathbf{w}_n - (T_{n-1}^{-1}T'_{n-1} + O(h_n))\mathbf{w}_{n-1}. \tag{4.12}$$

Therefore, the variable transformation term $T^{-1}T'\mathbf{w}$ is discretized explicitly at $n-1$ instead of at n. Usually the term $T^{-1}AT$ dominates, accounting for the practical success of the backward Euler and higher-order BDF schemes.

Our Case 3 corresponds to the domination of $T^{-1}AT$ in (4.11), (4.12): It is easy to see that the frozen coefficient part of the EUODE (2.20) is preserved in various transformations even after discretization (i.e., it is not significant whether the reformulation precedes discretization or vice versa). Therefore, a BDF discretization of any of the three formulations in this case results in a stiffly stable numerical method and performs well. The method of reduction to index 2 without the extra differentiation is most straightforward under these circumstances.

Case 4. In contrast to Case 3 above, the variable coefficient part of the various transformations, i.e., those terms involving derivatives of R, S, C, etc. in (2.13), (2.20), (3.5), and (3.17), does not generally get reproduced under discretization, as we saw in §4.1. The phenomenon is similar to that in (4.12), but it may be practically worse because unlike in the ODE case, R and S (and $\tilde{R}$) do not depend on A_j of (2.3) at all, so it is easy to envision situations where the variable coefficients part of the EUODE dominates. In such circumstances a backward Euler discretization may behave like a nonstiff discretization, causing a possible slowdown in an automatic integrator. Application of a state-space form method may be advantageous, then, if the restart difficulty is not present.

Example 2. Consider for $0 \le t \le 1$

$$\begin{aligned} x_1' &= (2-t)\nu y + q_1(t), \\ x_2' &= (\nu - 1)y + q_2(t), \\ 0 &= (t+2)x_1 + (t^2-4)x_2 + r(t), \end{aligned}$$

with $x_1(0) = 1$. Here $\nu \ge 1$ is a parameter. The inhomogeneities $\mathbf{q}$ and r are chosen to be

$$\begin{aligned} \mathbf{q} &= \begin{pmatrix} (1+\nu)e^t \\ (1+\frac{\nu-1}{2-t})e^t \end{pmatrix}, \\ r &= -(t^2+t-2)e^t, \end{aligned}$$

such that the exact solution is $x_1 = x_2 = e^t$, $y = -(e^t/(2-t))$.

This is essentially the same example as Example 1 in [1], but with $A_1 = 0$, so there is no frozen coefficient part in the EUODE. With

$$R(t) = \nu^{-1}(1-\nu, (2-t)\nu),$$

we have

$$\begin{pmatrix} R \\ C \end{pmatrix}^{-1} = (4-t^2)^{-1} \begin{pmatrix} (4-t^2)\nu & (2-t)\nu \\ (t+2)\nu & \nu - 1 \end{pmatrix},$$

so (2.5), (2.6), and (2.7) are satisfied with $K_1 = O(1)$, $K_2 = O(\nu)$, $K_3 = O(1)$. The EUODE for the homogeneous problem is

$$u' = R'Su = -\frac{\nu}{2-t}u$$

subject to an initial condition. Hence this is a stable problem with $K_4 = O(1)$, $K = O(\nu)$ in (2.15). Note also that $||F|| = O(\nu)$ and $||C'|| = O(1)$.

We certainly expect any of the numerical methods mentioned in this section to work well when the discretization step size $h = \max_n h_n$ satisfies $h\nu << 1$. It is more interesting to find out what happens, say, when $h\nu = 10$, which for $\nu = 1000$ yields a rather small step relative to the variation of the solution.

First consider a state-space reduction using $\tilde{R} = (1, 0)$. Thus $\tilde{S}^T = (1, 1/(2-t))$ and the homogeneous part of the ODE (3.17) is

$$\tilde{u}' = -\tilde{R}FC'\tilde{S}\tilde{u} = -\frac{\nu}{2-t}\tilde{u}.$$

If the inhomogeneous version of this ODE is discretized by a BDF scheme, or any other L-stable scheme, then not only is stability maintained for $h\nu$ large, but also accuracy *improves* as ν increases with h fixed, because there is only a fast, stable solution mode present and no slow ones. The transformation back from $\tilde{u}$ to $\mathbf{x}$ preserves this accuracy. Thus, the state-space reduction performs superbly here.

In contrast, the same BDF discretization applied to the other formulations has a significant nonstiff behavior. In Table 4.2 we display results using Baumgarte's technique with backward Euler and applying backward Euler directly to the original index-2 DAE. Tests are performed with $\nu = 1000$, $h = .01$.

TABLE 4.2
Behavior of methods for example 2.

Method	γ	Error	Drift
Baumgarte	0.	.19e−2	.85e−2
Baumgarte	1.	.22e−2	.49e−2
Baumgarte	10.	.10e−2	.29e−3
Baumgarte	100.	.27e−4	.93e−8
Baumgarte	1000.	.13e+42	.45e+39
Baumgarte	∞	.92e+74	.45e+58
Projected invariant	NA	.14e−4	0
State-space form ($\tilde{R} = (1, 0)$)	NA	.14e−4	0

A comparison between Tables 4.1 and 4.2 confirms that the practical control of the Baumgarte parameter may indeed be a nontrivial affair. Note also the excellent performance of the discretization of (4.7). □

Example 3. This example is a linear model in the form of a "mechanical system." We consider the initial value problem for the system

$$\begin{aligned} &(4.13a) & \mathbf{p}' &= \mathbf{v}, \\ &(4.13b) & M\mathbf{v}' &= \mathbf{f}(\mathbf{p},\mathbf{v},t) - G^T\lambda, \\ &(4.13c) & \mathbf{0} &= \mathbf{g}(\mathbf{p},t), \end{aligned}$$

with $\mathbf{g} = C(t)\mathbf{p} + r(t)$, $G = C$, and where $M(t)$ is symmetric positive definite. For $0 \le t \le 1$ we choose

$$\mathbf{f} = \begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{(2+t)\nu} \end{pmatrix} \mathbf{v} + \mathbf{q}(t), \qquad C = (1, t-2),$$

$$M(t) = \frac{1}{(2+t)\nu^2} \begin{pmatrix} \frac{\nu^2+(\nu-1)^2}{2-t} & -\nu(2\nu-1) \\ -\nu(2\nu-1) & 2(2-t)\nu^2 \end{pmatrix},$$

resulting in

$$B = M^{-1}C^T = \begin{pmatrix} (4-t^2)\nu \\ (\nu-1)(t+2) \end{pmatrix}.$$

The inhomogeneous terms $\mathbf{q}(t)$ and $r(t)$ and the initial conditions have been chosen so that the solutions for both components of $\mathbf{p}$ and $\mathbf{v}$ are e^t, and $\lambda = e^t/(2-t)$. This example is closely related to Example 2, and in particular, it has the same term $R'S$ with $R'' = 0$. We will consider its solution in two different formulations. In the first formulation, the twice-differentiated constraint is used to eliminate λ, and then the original constraint is reintroduced via a new Lagrange multiplier μ, to obtain

$$\begin{aligned} &(4.14a) & \mathbf{p}' &= \mathbf{v} + D\mu, \\ &(4.14b) & \mathbf{v}' &= HM^{-1}\mathbf{f} - Fz, \\ &(4.14c) & \mathbf{0} &= \mathbf{g}(\mathbf{p},t), \end{aligned}$$

where $z = 2C'\mathbf{v} + C''\mathbf{p} + r''$. This is the projected invariant formulation (3.27) with $m = 2, k = 1$. We will consider various choices for the projection matrix $D(t)$, satisfying that CD is nonsingular for each t. The second formulation is the following stabilized index-2 system,

$$\begin{aligned} &(4.15a) & \mathbf{p}' &= \mathbf{v} + D\mu, \\ &(4.15b) & M\mathbf{v}' &= \mathbf{f} - G^T\lambda, \\ &(4.15c) & \mathbf{0} &= \mathbf{g}, \\ &(4.15d) & \mathbf{0} &= \mathbf{g}' \equiv G\mathbf{v} + \mathbf{g}_t \end{aligned}$$

(in our linear case $\mathbf{g}' = C\mathbf{v} + C'\mathbf{p} + r'$). The latter formulation was proposed in [13] for $D = G^T$ $(= C^T)$.

In Table 4.3 we present the results for the projected invariant formulation (4.14) with projections D given by B, C^T, "unit" $= (0,1)^T$, and for the direct index-two ("d-2") formulation (i.e., where the constraints in (4.13) have been simply differentiated once), for values of $\nu = 1, 100$ and 1000. In Table 4.4 we present the results for the stabilized index-2 formulation (4.15) under the same conditions. All test results are with the backward Euler scheme on the interval $[0, 1]$, with a uniform step size $h = .01$. The recorded errors are measured at $t = 1$ in the indicated variable (maximum over the two components). For the first nine rows of Table 4.3 (and Tables 4.5 and 4.7 as well), "Drift" indicates the drift in the velocity constraint (derivative of the original constraint) at the endpoint of the interval. For direct index-2 formulation, where the drift in velocity constraint is 0 but the drift in position constraint is not, the latter is indicated. Since the drifts for the stabilized formulation (4.15) are essentially zero (except when everything blows up), they are not recorded in Table 4.4.

TABLE 4.3
Example 3, projected invariant formulation.

Projection	ν	μ	Error(p)	Error(v)	Drift
B	1	.16e−2	.13e−1	.12e−1	.49e−2
B	100	.91e−4	.38e−2	.35e−2	.14e−3
B	1000	.34e+73	.93e+74	.36e−2	.93e+74
C^T	1	.16e−2	.12e−1	.12e−1	.35e−2
C^T	100	.18e−2	.74e−2	.35e−2	.37e−2
C^T	1000	.18e−2	.72e−2	.36e−2	.37e−2
unit	1	.24e−2	.11e−1	.12e−1	.26e−2
unit	100	.27e−2	.65e−2	.35e−2	.29e−2
unit	1000	.27e−2	.64e−2	.36e−2	.29e−2
d−2	1	NA	.13e−1	.12e−1	.37e−2
d−2	100	NA	.37e−2	.17e−1	.36e−2
d−2	1000	NA	.10e+71	.11e+73	.10e+69

TABLE 4.4
Example 3, stabilized index-2 formulation.

Projection	ν	μ	Error(p)	Error(v)
B	1	.86e−4	.13e−1	.12e−1
B	100	.44e−6	.37e−2	.17e−1
B	1000	.28e−3	.35e−2	.78
C^T	1	.13e−3	.12e−1	.12e−1
C^T	100	.18e−7	.37e−2	.17e−1
C^T	1000	.99e+73	.18e+74	.20e+76
unit	1	.26e−3	.11e−1	.12e−1
unit	100	.42e−7	.38e−2	.17e−1
unit	1000	.16e+75	.15e+75	.16e+77

Additional experiments were carried out for the problem (4.13) with $\mathbf{f} = \mathbf{0}$ and the same exact solution. The results are summarized in Tables 4.5 and 4.6, which are analogous to Tables 4.3 and 4.4, respectively.

We note that, as predicted, methods using the B-projection or involving B in a Hessenberg index-2 formulation (even when stabilized using other projections) can

TABLE 4.5
Example 3 with $\mathbf{f} = \mathbf{0}$, projected invariant formulation.

Projection	ν	μ	Error(p)	Error(v)	Drift
B	1	.16e−2	.12e−1	.86e−2	.49e−2
B	100	.91e−4	.38e−2	.35e−2	.14e−3
B	1000	.34e+73	.93e+74	.35e−2	.93e+74
C^T	1	.17e−2	.11e−1	.86e−2	.35e−2
C^T	100	.18e−2	.73e−2	.35e−2	.37e−2
C^T	1000	.81e−2	.72e−2	.35e−2	.37e−2
unit	1	.25e−2	.10e−1	.86e−2	.27e−2
unit	100	.27e−2	.65e−2	.35e−2	.28e−2
unit	1000	.27e−2	.64e−2	.35e−2	.28e−2
d-2	1	NA	.12e−1	.86e−2	.37e−2
d-2	100	NA	.62e−2	.13e−1	.36e−2
d-2	1000	NA	.60e−2	.13e−1	.36e−2

TABLE 4.6
Example 3 with $\mathbf{f} = \mathbf{0}$, stabilized index-2 formulation.

Projection	ν	μ	Error(p)	Error(v)
B	1	.74e−4	.12e−1	.86e−2
B	100	.45e−4	.18	.19
B	1000	.11e+150	.32e+150	.28e+153
C^T	1	.11e−3	.11e−1	.86e−2
C^T	100	.34e−4	.53e−2	.12e−1
C^T	1000	.33e−4	.52e−2	.12e−1
unit	1	.22e−3	.99e−2	.86e−2
unit	100	.68e−4	.50e−2	.12e−1
unit	1000	.67e−4	.49e−2	.12e−1

experience a serious error growth when $h\nu$ is large, due to the large size of the $R'S$-term. Only the projected invariant formulations using the "good" projections C^T and "unit" yield acceptable results for both choices of $\mathbf{f}$ when $h\nu = 10$. The backward Euler scheme performs like a nonstiff integrator in these circumstances for the other methods. The good behavior of the projected invariant formulation for the projections C^T and "unit" follows directly from the discussion earlier in this section.

Let us calculate the EUODE for this example. Writing (4.13) with $\mathbf{f} = E(t)\mathbf{v}$ in the form (2.3), we have $m \leftarrow 2$, $\mathbf{x} \leftarrow \mathbf{p}$, $\mathbf{y} \leftarrow -\lambda$, $A_1 \leftarrow 0$, $A_2 \leftarrow M^{-1}E$. With R and S as in Example 2 we obtain

$$(RA_1 + R'')S\mathbf{u} = 0,$$

$$(RA_2 + 2R')(S\mathbf{u})' = \nu[RM^{-1}E + 2(0,-1)][Su' + S'u].$$

For E corresponding to Tables 4.3 and 4.4, $RM^{-1}E = (0,1)$, so by (2.13) the homogeneous EUODE is

$$u'' = -\frac{\nu}{2-t}u' - \frac{\nu}{(2-t)^2}u.$$

For $E = 0$ corresponding to Tables 4.5 and 4.6, the homogeneous EUODE is

$$u'' = -\frac{2\nu}{2-t}u' - \frac{2\nu}{(2-t)^2}u.$$

So the EUODE in both cases is stable for $\nu > 0$ and stiff for $\nu >> 1$.

If we now choose

$$E = \begin{pmatrix} 0 & 0 \\ 0 & \frac{2}{(2+t)\nu} \end{pmatrix},$$

then $RA_2 = -2R'$, so the EUODE is nonstiff. In Tables 4.7 and 4.8 we record results analogous to Tables 4.3 and 4.4 for this case where the EUODE is nonstiff.

TABLE 4.7
Example 3 with a nonstiff EUODE, projected invariant formulation.

Projection	ν	μ	Error(p)	Error(v)	Drift
B	1	.15e−2	.15e−1	.18e−1	.49e−2
B	100	.88e−4	.10e−1	.18e−1	.32e−3
B	1000	.34e+73	.93e+74	.18e−1	.93e+74
C^T	1	.15e−2	.13e−1	.18e−1	.34e−2
C^T	100	.15e−2	.13e−1	.18e−1	.34e−2
C^T	1000	.15e−2	.13e−1	.18e−1	.34e−2
unit	1	.22e−2	.12e−1	.18e−1	.26e−2
unit	100	.22e−2	.12e−1	.18e−1	.26e−2
unit	1000	.22e−2	.12e−1	.18e−1	.26e−2
d−2	1	NA	.14e−1	.18e−1	.37e−2
d−2	100	NA	.30e+2	.30e+15	.30
d−2	1000	NA	.17	.54	.53e−2

TABLE 4.8
Example 3 with a nonstiff EUODE, stabilized index-2 formulation.

Projection	ν	μ	Error(p)	Error(v)
B	1	.11e−3	.15e−1	.18e−1
B	100	.47e−4	.37	.35
B	1000	.54e−6	.36	.34
C^T	1	.16e−3	.13e−1	.18e−1
C^T	100	.11e+2	.23	.22e+4
C^T	1000	.18e−2	.60e−1	.34
unit	1	.32e−3	.12e−1	.18e−1
unit	100	.13e+4	.14	.13e+6
unit	1000	.28e−2	.26e−1	.27

We note with no surprise that the problem does not get easier when the large terms in the EUODE cancel one another. The results in Table 4.7 for the projections with C^T and "unit" are independent of ν: this is because (4.14b) is independent of ν in this special case. □

Example 4 [2]. This example is also in the "mechanical system form" (4.13). This time we choose $M = I$ but let the constraint matrix vary possibly rapidly, as in Example 1. We set

$$\mathbf{g} = C(t)\mathbf{p} + r(t), \quad G = C = (\sin(\nu t), \cos(\nu t)) = B^T, \quad \mathbf{f} = -\nu^2\mathbf{v} + \mathbf{q}(t)$$

and choose initial conditions and inhomogeneities such that the solution is the same as in Example 3. The stability of this problem is discussed in [2].

Here we test the performance of two projected invariant formulations, in addition to the previous reformulations (4.14) and (4.15) (with $D = G^T$). In both of these additional formulations we require satisfaction not only of the "position constraints" (4.15c), but also of their derivative (4.15d) ("velocity constraints"). The first of these additional formulations is as in §3.5 with $k = m = 2$ and $B_1 = B_2 = C^T$. This gives the system

$$\mathbf{p}' = \mathbf{v} + G^T\mu, \tag{4.16a}$$
$$\mathbf{v}' = HM^{-1}\mathbf{f} - Fz + G^T\tau, \tag{4.16b}$$
$$\mathbf{0} = \mathbf{g}(\mathbf{p}, t), \tag{4.16c}$$
$$\mathbf{0} = \mathbf{g}' \equiv G\mathbf{v} + \mathbf{g}_t. \tag{4.16d}$$

The method (4.16) should certainly perform well for Example 3, but here the matrix function $L \equiv (\partial\mathbf{g}'/\partial\mathbf{p})$ $(= C')$ may contain large elements when ν is large. The second variant balances this out by adding another stabilizing term to (4.16a), replacing it by

$$\mathbf{p}' = \mathbf{v} + G^T\mu + L^T\tau. \tag{4.17}$$

In Table 4.9 we compare numerical results for various values of ν and h using a backward Euler discretization of (i) the previous, simpler projected invariant formulation (4.14) with $D = G^T$ (denoted "Pa"); (ii) the projected invariant method (4.16) (denoted "Pb"); (iii) the projected invariant method (4.17), (4.16b), (4.16c), (4.16d) (denoted "Pc"); and (iv) the stabilized index reduction method (4.15) with $D = G^T$ [13] (denoted "S").

Note that the various variants perform qualitatively similarly when $h\nu$ is small. But when $h\nu = 10$, there are large errors in $\mathbf{v}$ for the formulations Pb and S (these two are almost identical for this problem). The projected invariant formulations Pa and Pc, which do balance the constraint matrix and the multiplier matrix in the index-2 formulation, perform significantly better for $h\nu$ large. The scheme Pc is more complicated than Pa, though, so computing and using L in (4.17) may not be desirable for reasons other than stiff stability. $\square$

A number of methods have been proposed in the literature (see [9] and references therein, [1], [12], [16], and [17]), where at each step in t, an integration step for the ODE (3.18a), (3.22) or another form of the DAE is followed by a projection using a weighted least squares norm to satisfy the constraints (3.25) at the end of the step. Thus, using, e.g., backward Euler for the unstabilized (4.14a) (i.e., with $D = 0$) and (4.14b), we have at the nth step

$$h^{-1}(\tilde{\mathbf{p}}_n - \mathbf{p}_{n-1}) = \mathbf{v}_n, \tag{4.18a}$$
$$h^{-1}(\mathbf{v}_n - \mathbf{v}_{n-1}) = H_nM_n^{-1}\mathbf{f}_n - F_n\mathbf{z}_n, \tag{4.18b}$$

and then we find $\mathbf{p}_n$, which satisfies

$$\mathbf{g}(\mathbf{p}_n, t_n) = \mathbf{0} \tag{4.19}$$

and minimizes

$$(\mathbf{p}_n - \tilde{\mathbf{p}}_n)^T W_n(\mathbf{p}_n - \tilde{\mathbf{p}}_n), \tag{4.20}$$

TABLE 4.9
Example 4, additional projected invariant formulations.

Projection	ν	h	Error(p)	Error(v)
Pa	1	.01	.22e−2	.82e−2
		.001	.22e−3	.81e−3
Pb		.01	.36e−2	.47e−2
		.001	.37e−3	.48e−3
Pc		.01	.29e−2	.30e−2
		.001	.28e−3	.30e−3
S		.01	.36e−2	.47e−2
		.001	.36e−3	.48e−3
Pa	100	.1	.22e−1	.17e−1
		.01	.19e−3	.65e−4
		.001	.59e−5	.15e−4
Pb		.1	.22e−1	.22e+1
		.01	.19e−3	.19e−1
		.001	.57e−5	.56e−3
Pc		.1	.17e−3	.17e−1
		.01	.68e−6	.66e−4
		.001	.16e−6	.15e−4
S		.1	.22e−1	.22e+1
		.01	.19e−3	.19e−1
		.001	.58e−5	.56e−3

where W_n is a symmetric positive definite matrix. This idea can clearly be written in the generality of §3.5 and gives another variant for resolving overdetermination.

The necessary conditions for the constrained minimization (4.19), (4.20) are

$$W_n(\mathbf{p}_n - \tilde{\mathbf{p}}_n) = C_n^T \mu_n, \tag{4.21}$$

where μ_n is a Lagrange multiplier. Therefore,

$$\mathbf{p}_n = \tilde{\mathbf{p}}_n + W_n^{-1} C_n^T \mu_n = \mathbf{p}_{n-1} + h\mathbf{v}_n + W_n^{-1} C_n^T \mu_n. \tag{4.22}$$

However, an unfortunate choice of W_n may again produce a nonstiff behavior out of a BDF scheme, because (4.22) is in essence a backward Euler discretization of (4.14a) with $D = W^{-1}C^T$. For Example 3, in particular, the choice $W = M$ is not advisable.

For some schemes, though, the choice of W is not sufficiently arbitrary. For instance, in [1] the integration step is an implicit Runge–Kutta (or collocation) step applied directly to an index-2 DAE (4.1). This necessitates in the following projection the choice of W so that $W^{-1}C^T = B$. Therefore, that method applied to Example 2 also behaves like a nonstiff integrator. A way to remedy this is to transform (4.1) into (4.7) before applying the projected Runge–Kutta method. For Example 2, this works very well.

In the context of mechanical systems, we note that the projected invariant scheme (4.14) satisfies precisely only the position constraints, not their derivative relation $\mathbf{g}' = \mathbf{0}$ (which gives velocity constraints). Should the latter be deemed important to satisfy precisely as well, it can be achieved with the formulation (4.14) as described above, i.e., by projecting $\mathbf{v}_n$ at the end of each step to satisfy (4.15d), minimizing the required change in $\mathbf{v}_n$ (in the least squares norm) and keeping $\mathbf{p}_n$ unchanged (cf. [2]).

5. Conclusions. We have considered various problem formulations and their discretizations for higher-order, higher-index DAEs such as those that arise in the numerical integration of Lagrange's equations of the first kind for multibody dynamics. A linearized form of the equations was considered, allowing a methodical examination of a number of methods that are in use in practice with respect to stability. While the numerical examples we use do not derive directly from particular mechanical systems, they are simple to follow thoroughly and at the same time they highlight effects that we believe occur in some actual mechanical systems. This yields a number of tentative conclusions, based on the methods considered.

1. All reasonable problem reformulations used in practice are stable under certain mild assumptions. The exception is a direct index reduction, which has an algebraic instability of degree $m-1$. Thus, for holonomic constraints, two direct constraint differentiations yield a linear instability. Applying only one direct differentiation is still stable, though not asymptotically stable. (Note, however, that asymptotic stability of $\mathbf{v}$ in (3.9b) does not yield a similar statement for $\mathbf{u}$ in (3.9a).)

2. Applying the same discretization to two stable problem formulations does not necessarily yield similar method characteristics.

3. For simple, slowly varying nonstiff problems, a number of good alternatives exist. One is a Baumgarte stabilization coupled with an explicit discretization. Another is the half-explicit methods; see, e.g., [17] and [5].

4. For problems with a stiff frozen coefficient part, a BDF (or other stiff) discretization applied to a stable index-2 reduction is recommended.

5. For heterogeneous problems, where the mass matrix has widely varying eigenvalues (or when C is much better behaved than B in (2.3)), and for problems with rapidly varying constraints, the projected invariants stabilization (4.14) (with the projection based on C^T) coupled with a BDF discretization or other suitable stiff method is recommended. If it is important that the velocity constraints be satisfied precisely, then we recommend either the projected invariants method Pa (4.14) followed by a projection onto the velocity constraints as described at the end of §4, or else the projected invariants method Pc (4.17), (4.16b), (4.16c), (4.16d).

REFERENCES

[1] U. Ascher and L. Petzold, *Projected implicit Runge–Kutta methods for differential-algebraic equations*, SIAM J. Numer. Anal., 28 (1991), pp. 1097–1120.

[2] ———, *Projected collocation for higher-order higher-index differential-algebraic equations*, Tech. Rep. 91-9, Dept. of Computer Science, Univ. of British Columbia, Vancouver, British Columbia, Canada, 1991.

[3] U. Ascher, R. Mattheij, and R. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[4] J. Baumgarte, *Stabilization of constraints and integrals of motion in dynamical systems*, Comp. Math. Appl. Mech. Engrg., 1 (1976), pp. 1–16.

[5] V. Brasey, *A half-explicit Runge–Kutta method of order* 5 *for solving constrained mechanical systems*, Tech. Rep., Dept. of Mathematics, Univ. de Geneve, Geneva, Switzerland, 1991.

[6] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland, Amsterdam, 1989.

[7] A. Bunse-Gerstner, R. Byers, V. Mehrmann, and N. Nichols, *Numerical computation of an analytic singular value decomposition of a matrix valued function*, Tech. Rep., Universität Bielefeld, Bielefeld, Germany, 1990.

[8] L. Dieci, M. Osborne, and R. Russell, *A Riccati transformation method for solving linear* BVP*s*: II. *Computational aspects*, SIAM J. Numer. Anal., 25 (1988), pp. 1074–1092.

[9] E. Eich, C. Führer, B. Leimkuhler, and S. Reich, *Stabilization and projection methods for multibody dynamics*, Res. Rep., Inst. Math., Helsinki Univ. of Technology, Helsinki, Finland, 1990.

[10] C. FÜHRER AND B. LEIMKUHLER, *Numerical solution of differential-algebraic equations for constrained mechanical motion*, Numer. Math., 59 (1991), pp. 55–69.

[11] C.W. GEAR, *Differential-algebraic equation index transformations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 39–47.

[12] ———, *Maintaining solution invariants in the numerical solution of* ODE*s*, SIAM J. Sci. Statist. Comp., 7 (1986), pp. 734–743.

[13] C.W. GEAR, G. GUPTA, AND B. LEIMKUHLER, *Automatic integration of the Euler–Lagrange equations with constraints*, J. Comput. Appl. Math., 12 (1985), pp. 77–90.

[14] H.-O. KREISS, *Difference methods for stiff ordinary differential equations*, SIAM J. Numer. Anal., 15 (1978), pp. 21–58.

[15] H.-O. KREISS, N.K. NICHOLS, AND D.L. BROWN, *Numerical methods for stiff two-point boundary value problems*, SIAM J. Numer. Anal., 23 (1986), pp. 325-368.

[16] CH. LUBICH, *h^2-Extrapolation methods for differential-algebraic systems of index* 2, IMPACT, 1 (1989), pp. 260–268.

[17] ———, *Extrapolation integrators for constrained multibody systems*, Tech. Rep., Universität Innsbruck, Innsbruck, Austria, 1990.

[18] F. A. POTRA AND W. C. RHEINBOLDT, *On the numerical solution of the Euler–Lagrange equations*, J. Mech. Structures Mach., to appear.

[19] W. SCHIEHLEN, ED., *Multibody Systems Handbook*, Springer-Verlag, Berlin, New York, 1990.

[20] R. A. WEHAGE AND E. J. HAUG, *Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems*, J. Mech. Design, 104 (1982), pp. 247–255.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 121–136, January 1993

REDUCING THE SYMMETRIC MATRIX EIGENVALUE PROBLEM TO MATRIX MULTIPLICATIONS*

SHING-TUNG YAU† AND YA YAN LU‡

Abstract. A numerical method for the symmetric matrix eigenvalue problem is developed by reducing it to a number of matrix–matrix multiplications. For matrices of size n, the number of such multiplications is on the order of $\log_2 n$. On high performance parallel computers, it is important to minimize memory reference, since the movement of data between memory and registers can be a dominant factor for the overall performance. The matrix–matrix multiplication is more efficient than matrix–vector or vector–vector operations, since it involves $O(n^3)$ floating point operations while creating only $O(n^2)$ data movements. The number of data movements of the traditional methods based on reduction to the tridiagonal form is $O(n^3)$, while that of our method is $O(n^2 \log_2 n)$. Asymptotically, there are fast numerical algorithms for matrix multiplications that require less than $O(n^3)$ floating point operations. One example is the $O(n^{2.376})$ method of Coppersmith and Winograd [*Proc. 19th Ann. ACM Symp. Theory Comput.*, 1987, pp. 1–6]. Therefore, in principle, our method for the symmetric matrix eigenvalue problem requires only $O(n^{2.376} \log_2 n)$ operations.

Key words. eigenvalue problem, matrix multiplication, parallel computation, FFT

AMS(MOS) subject classifications. 65F15, 65N25, 15A18

1. Introduction.

Recently, much effort has been devoted to developing efficient numerical algorithms for matrix eigenvalue problems on high performance parallel computers. New methods and modifications of the traditional methods [27], [18], [24] developed on sequential computers are being studied in the new environment. The Jacobi method [14] has attracted much attention [22], [4], since it is natural for parallelization. Meanwhile, the idea that one first reduce the matrix to a condensed form through orthogonal similarity transform is still the basis for many studies. The reduction process itself has been analyzed and the resulting block methods [3], [23], [7], [12] could have a better performance on parallel computers. When the matrix is symmetric, new methods have been developed for the condensed form: tridiagonal matrices. The method in [17], [13], and [15] is based on evaluating the characteristic polynomial of the tridiagonal matrix. It computes the eigenvalues through multisectioning and the eigenvectors through inverse iteration. Methods based on the idea of divide and conquer have been developed and implemented in [6] and [9]. Recursively, two smaller tridiagonal eigenvalue problems are first solved, then combined to obtain the eigensystem of the larger tridiagonal matrix.

One important issue that appeared in the high-performance computers is the cost of data movement. It becomes necessary to organize the algorithms such that we reuse the data as much as possible. Matrix–matrix multiplication performs $O(n^3)$ operations (without using any asymptotically faster algorithms) on $O(n^2)$ data, and it is more efficient than matrix–vector or vector–vector operations. Therefore, it is desirable to develop algorithms that are rich in matrix–matrix operations. This has led to the development of block algorithms for matrix computations [10]. These block algorithms are usually variants of the original well-established sequential algorithms. We believe that it is helpful to think about the problem from a new starting point and to

*Received by the editors December 18, 1990; accepted for publication (in revised form) December 23, 1991.

†Department of Mathematics, Harvard University, Cambridge, Massachusetts 02138.

‡Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York 12180.
The research of this author was supported by National Science Foundation/Defense Advanced Research Projects Agency grant DMS 87-11394.

develop new algorithms that fully take advantage of the new computer architectures.

In this paper, we develop a method that reduces the symmetric matrix eigenvalue problem to a number of matrix multiplications. The total number of such multiplications is $O(\log_2 n)$. The traditional algorithm performs $O(n^3)$ floating point operations requiring $O(n^3)$ data movement. Without using any asymptotically fast matrix multiplication algorithms, our method performs $O(n^3 \log_2 n)$ operations with $O(n^2 \log_2 n)$ data movement. Meanwhile, much effort has been devoted to achieving a lower asymptotic bound [19], [25] for matrix multiplications. The method in [5] requires only $O(n^{2.376})$ operations to multiply two $n \times n$ matrices. Practical implementation of the Strassen method [25] that requires $O(n^{\log_2 7})$ operations on the CRAY-2 has been reported [2]. In any circumstance, from a theoretical point of view, the number of operations required in our algorithm has a lower asymptotic bound: $O(n^{2.376} \log_2 n)$.

2. Basic ideas. Consider a real symmetric matrix A with eigenvalues λ_1, λ_2, $\ldots$, λ_n and eigenvectors g_1, g_2, $\ldots$, g_n. Starting from an initial vector expanded in terms of the eigenvectors as

$$v_0 = \alpha_1 g_1 + \alpha_2 g_2 + \cdots + \alpha_n g_n,$$

we can construct a polynomial of A, say $P_N(A)$, and multiply it by the vector v_0. We obtain

$$P_N(A)v_0 = \alpha_1 P_N(\lambda_1) g_1 + \alpha_2 P_N(\lambda_2) g_2 + \cdots + \alpha_n P_N(\lambda_n) g_n.$$

The central idea in the polynomial acceleration methods, such as those for a linear system of equations, is to choose the polynomial so that it peaks up at one of the eigenvalues and is close to zero otherwise. The vector $P_N(A)v_0$ can thus be regarded as an approximation to the corresponding eigenvector.

There are several difficulties with this approach. It apparently only computes one eigenpair at a time. Meanwhile, the initial peak of the polynomial may not be so close to the desired eigenvalue that the computed vector is not accurate enough, and we need to shift the peak of $P_N(A)$ and repeat the process. All of these are too expensive to be of any practical interest.

The main purpose of this paper is to introduce the fast Fourier transform (FFT) into this process, such that the above difficulties can be overcome and all the eigenvalues of A can be computed simultaneously.

Consider a unitary matrix B whose eigenvalues $\mu_1, \mu_2, \ldots, \mu_n$ all lie on the unit circle. Let $P_N(z)$ be a polynomial that has a peak at $z = 1$ and is constructed to be close to zero on the unit circle away from the vicinity of $z = 1$. Let the eigenvectors of B be $g_1, g_2, \ldots, g_n$ (the same notations that are used for the eigenvectors of A) and the initial vector v_0 be expanded as $v_0 = \sum_{j=1}^{n} \alpha_j g_j$. We have that

$$u(\lambda) = P_N(e^{-i\lambda}B)v_0 = \sum_{j=1}^{n} \alpha_j P_N(e^{-i\lambda}\mu_j) g_j.$$

Apparently, if λ is chosen such that μ_{j_0} is close to $e^{i\lambda}$ and other eigenvalues of B are not "close" to $e^{i\lambda}$, then the coefficients of g_j will be small except when $j = j_0$. Thus $u(\lambda)$ can be regarded as an approximation to the eigenvector of μ_{j_0}. If the polynomial $P_N(z)$ is written as

$$P_N(z) = \sum_{j=0}^{N-1} \beta_j z^j,$$

then we have that

$$u(\lambda) = \sum_{j=0}^{N-1} \beta_j B^j v_0 e^{-ij\lambda}$$

$$= \sum_{j=0}^{N-1} \beta_j v_j e^{-ij\lambda},$$

where $v_j = B^j v_0$. Therefore, if the vectors v_j are computed first, FFT can be used to compute $u(\lambda)$ at many different values of λ simultaneously.

In order to apply the above approach to the symmetric matrix eigenvalue problem, we need to first link the matrix A with a unitary matrix, then find a suitable polynomial $P_N(z)$ and finally compute the vectors v_j. After the vectors $u(\lambda)$ are computed through FFT, we need to select the vectors that can be regarded as eigenvectors. When there are close or multiple eigenvalues, a local refined scheme with possible supplementary vectors is proposed such that all the eigenvalues can be computed to the full double precision. We discuss these issues in the next sections.

3. Unitary matrix. The natural choice for the unitary matrix B is e^{iA}. The fact that A is a symmetric matrix brings an important difference between e^{iA} and e^{A}. The eigenvalues of e^{A} may be extended to a large range such that the finite precision arithmetic computers may regard e^{A} as near rank deficient. The situation for e^{iA} is different, since all the eigenvalues have norm one.

However, we still want to avoid the situation where all the eigenvalues of e^{iA} lie in just part of the unit circle. We prefer a more or less uniform distribution on the unit circle for more efficiency of our algorithm. Therefore, we first estimate the largest and smallest eigenvalues of A, and find a lower bound and an upper bound for the spectrum. Then, the matrix is shifted and scaled (still denoted by A) such that its eigenvalues are within the interval $[0, 2\pi)$.

Estimating the extreme eigenvalues of A is not a difficult task, since we do not really need any high precision for the extreme eigenvalues—only the true lower and upper bounds are needed. A few steps of the Lanczos method are usually sufficient for this purpose. A small number of matrix–vector multiplications are needed in this step.

The computation of e^{iA} is an expensive step. On sequential computers, this could be a more difficult task than the symmetric matrix eigenvalue problem itself. However, the situation on high-performance parallel computers is different, since its computation can be carried out in a small number of efficient matrix–matrix multiplications. The following method requires only six such multiplications to compute $\cos(A)$ with double precision. The expansion for $\sin(A)$ is also given, although we only use it to compute $\sin(A)v_0$, where v_0 is the initial vector. (If $\sin(A)$ is desired, we can compute it in two more matrix multiplications.) The method is based on the Chebyshev expansions of $\cos(\pi x)$ and $\sin(\pi x)/x$ for x within $[-1, 1]$ and it essentially uses the method for evaluating polynomials of a matrix proposed by Paterson and Stockmeyer [20] and Van Loan [26].

For $\cos(\pi x)$, we have

$$\cos(\pi x) = \tilde{c}_0 + \tilde{c}_1 T_2(x) + \tilde{c}_2 T_4(x) + \cdots + \tilde{c}_{10} T_{20}(x) + \cdots.$$

No terms after T_{20} are needed for double precision. Using the formula $T_{j+k} = 2T_j T_k - T_{j-k}$ (for $j > k$), we can rewrite the above expansion in terms of $T_0, T_2, \ldots, T_{10}$. A

TABLE 1
Coefficients for $\cos(\pi x)$ *and* $\sin(\pi x)/x$.

k	c_k	s_k
0	-0.30424217764410000D+00	0.13475263146739893D+01
1	-0.97086786526208879D+00	-0.15565912566288138D+01
2	0.30284915514918853D+00	0.22275791179570700D+00
3	-0.29091923138480781D-01	-0.14193172035956216D-01
4	0.13914657144750502D-02	0.51171426651805082D-03
5	-0.40189944510754951D-04	-0.11893504653342084D-04
6	0.15565534023630612D-05	0.38601607212752680D-06
7	-0.21653060683708492D-07	-0.46251621156060299D-08
8	0.22702183556992598D-09	0.42608348743303411D-10
9	-0.18590593629629630D-11	-0.31125835925925928D-12
10	0.12222734929968973D-13	0.18474722480287380D-14

similar expansion is developed for $\sin(\pi x)$ based on the even function $\sin(\pi x)/x$. We have

$$\cos(\pi x) \approx c_0 + c_1T_2 + c_2T_4 + \cdots + c_5T_{10} + T_{10}(c_6T_2 + c_7T_4 + \cdots + c_{10}T_{10}),$$
$$\sin(\pi x)/x \approx s_0 + s_1T_2 + s_2T_4 + \cdots + s_5T_{10} + T_{10}(s_6T_2 + s_7T_4 + \cdots + s_{10}T_{10}).$$

For the symmetric matrix A whose eigenvalues are within $[0, 2\pi)$, we define $X = A/\pi - I$. Therefore, $e^{iA} = -e^{i\pi X}$. Using the formula $T_{j+k} = 2T_jT_k - T_{j-k}$, we can compute $T_2(X), T_4(X), T_6(X), T_8(X)$, and $T_{10}(X)$ in a total of five matrix multiplications, $\cos(\pi X)$ in one more multiplication. However, $\sin(\pi X)$ requires two more multiplications, since the expansion used is for the even function $\sin(\pi x)/x$. Fortunately, only the multiplication of $\sin(\pi X)$ with a given initial vector is really needed, which can be computed by the above expansion with only matrix–vector multiplications involved. Therefore, a total of six multiplications each involving two real symmetric matrices are needed. We list the coefficients in Table 1.

4. Chebyshev polynomial. We consider a polynomial of degree $N-1$, $P_N(z)$, such that its real part has a peak at $z = 1$ and is small on the unit circle except at the vicinity of $z = 1$. We can scale the value of $P_N(z)$ at $z = 1$. This leads to a constrained uniform approximation problem: find $\beta_0, \beta_1, \beta_2, \ldots, \beta_{N-1}$, such that the polynomial $P_N(z) = \sum_{k=0}^{N} \beta_k z^k$ satisfies the following two conditions:

1. $P_N(1) = 1$;
2. The maximum of $|Re(P_N(z))|$ over the part of the unit circle defined as

$$S_\delta^1 = \{z = e^{i\theta} | \delta\pi \le \theta \le (2-\delta)\pi\}$$

is minimized, where δ is a small number that we can choose.

It turns out that the coefficients $\beta_0, \beta_1, \ldots, \beta_{N-1}$ can be taken as real numbers because of the symmetry with respect to the real axis on the complex z plane.

Therefore,

$$\begin{aligned} Re(P_N(z)) &= \beta_0 + \beta_1 \cos(\theta) + \cdots + \beta_{N-1} \cos((N-1)\theta) \\ &= \beta_0 + \beta_1 T_1(\cos(\theta)) + \cdots + \beta_{N-1} T_{N-1}(\cos(\theta)) \\ &= Q_N(\cos(\theta)). \end{aligned}$$

The polynomial $Q_N(x)$ defined above satisfies the conditions: (i) $Q_N(1) = 1$; and (ii) $\max |Q_N(x)|$ over $-1 \le x \le \cos(\delta\pi)$ is minimized. Therefore, $Q_N(x)$ is the Chebyshev polynomial [21] translated to the interval $[-1, \cos(\delta\pi)]$ and scaled with a constant. That is,

$$Q_N(x) = \frac{T_{N-1}\left(\dfrac{2x + 1 - \cos(\delta\pi)}{1 + \cos(\delta\pi)}\right)}{T_{N-1}\left(\dfrac{3 - \cos(\delta\pi)}{1 + \cos(\delta\pi)}\right)}.$$

The coefficients $\beta_0, \beta_1, \ldots, \beta_{N-1}$ are simply the Chebyshev expansion coefficients of $Q_N(x)$,

$$Q_N(x) = \beta_0 T_0(x) + \beta_1 T_1(x) + \cdots + \beta_{N-1} T_{N-1}(x).$$

In order to compute these coefficients, we define $\beta_j^{(k)}$ by

$$T_k(ax + b) = \sum_{j=0}^{k} \beta_j^{(k)} T_k(x),$$

for $k = 0, 1, \ldots, N-1$, where

$$a = \frac{2}{1 + \cos(\delta\pi)}, \qquad b = \frac{1 - \cos(\delta\pi)}{1 + \cos(\delta\pi)}.$$

The first two sets of coefficients are

$$\beta_0^{(0)} = 1; \quad \beta_0^{(1)} = b, \quad \beta_1^1 = a.$$

Let $\beta_j^{(k)} = 0$; if $j > k$, we have the following recurrence relationships:

$$\beta_j^{(k+1)} = a\left(\beta_{j-1}^{(k)} + \beta_{j+1}^{(k)}\right) + 2b\beta_j^{(k)} - \beta_j^{(k-1)}$$

for $j > 1$, and

$$\beta_0^{(k+1)} = a\beta_1^{(k)} + 2b\beta_0^{(k)} - \beta_0^{(k-1)},$$

$$\beta_1^{(k+1)} = a\left(2\beta_0^{(k)} + \beta_2^{(k)}\right) + 2b\beta_0^{(k)} - \beta_0^{(k-1)}.$$

These recurrence relationships are used to compute the coefficients $\beta_j^{(k)}$ for increasing values of k. After $\beta_0^{(N-1)}, \beta_1^{(N-1)}, \ldots, \beta_{N-1}^{(N-1)}$ are computed, we simply multiply by $\kappa = 1/T_{N-1}((3 - \cos\delta\pi)/(1 + \cos\delta\pi))$ to obtain $\beta_0, \beta_1, \ldots, \beta_{N-1}$.

The total number of floating point operations involved in this step is $O(N^2)$. Meanwhile, it is efficient on vector and parallel computers because of its highly vectorizable nature.

5. v_j and $u(\lambda)$. We consider the computation of the vectors $v_j = e^{ijA}v_0$ for $j = 1, 2, \ldots, N-1$. The straightforward approach requires $N - 1$ matrix–vector multiplications. While v_0 is the initial vector, $v_1 = e^{iA}v_0$ can be computed easily. Since we have computed $\cos(A)$, we are ready to obtain the real part of v_1. The expansion developed for $\sin(A)$ can be used to compute $\sin(A)v_0$ in just a few matrix–vector multiplications. The remaining vectors can be obtained through the recursive formula

$$v_{j+1} = 2\cos(A)v_j - v_{j-1}$$

for $j \geq 1$. N is usually taken as $O(n)$, therefore, computing all v_j (for $j = 1, 2, \ldots, N-1$) requires a similar amount of operations as the orthogonal reduction of a symmetric matrix to a tridiagonal matrix used in the standard method. Because of the efficiency of matrix–matrix operations, we reformulate the algorithm as follows:

$$\begin{aligned}
&\text{(a)} \quad B_1 = \cos(A),\\
&\qquad\; v_2 = 2B_1v_1 - \overline{v}_0;\\
&\text{(b)} \quad B_2 = \cos(2A),\\
&\qquad\; (v_3, v_4) = 2B_2(v_1, v_2) - \overline{(v_1, v_0)};\\
&\text{(c)} \quad B_4 = \cos(4A),\\
&\qquad\; (v_5, v_6, v_7, v_8) = 2B_4(v_1, v_2, v_3, v_4) - \overline{(v_3, v_2, v_1, v_0)};\\
&\text{(d)} \quad \ldots\ldots\ldots\ldots
\end{aligned}$$

In $\log_2 N$ steps, all these vectors are computed.

Once these vectors are computed, they are used to compute the vectors $\{u(\lambda)\}$, which can be kept real. We have

$$\begin{aligned}
u(\lambda) &= Re \sum_{j=0}^{N-1} \beta_j v_j e^{-ij\lambda}\\
&= \sum_{j=0}^{N-1} \beta_j Re(v_j)\cos(j\lambda) + \beta_j Im(v_j)\sin(j\lambda).
\end{aligned}$$

The vectors $v_0, v_1, \ldots, v_{N-1}$ are complex (except that the initial v_0 can be taken as real). Therefore, we can have a FFT program that computes real vectors $u(\lambda)$ at $2N$ different values of λ and overwrites the memory space of $v_0, v_1, \ldots, v_{N-1}$. For $l = 0, 1, \ldots, 2N-1$, we denote $u(\lambda)|_{\lambda = l\pi/N}$ by u_l, that is,

$$u_l = Re \sum_{j=0}^{N-1} \beta_j e^{-ijl\pi/N} v_j.$$

When N is taken as a power of 2, the resulting FFT is quite efficient.

6. Selection and refinement. We have now computed the vectors

$$\begin{aligned}
u_l &= Re P_N(e^{-il\pi/N}e^{iA})v_0\\
&= \sum_{k=1}^{n} \alpha_k P_N(e^{i(\lambda_k - l\pi/N)})g_k
\end{aligned}$$

for $l = 0, 1, \ldots, 2N-1$, where $P_N(z) = \sum_{j=0}^{N-1} \beta_j z^j$ is the previously defined polynomial; λ_k, g_k are the eigenvalues and eigenvectors of matrix A, respectively; and the initial vector v_0 (normalized) is expanded as

$$v_0 = \sum_{k=1}^{n} \alpha_k g_k.$$

The purpose of this section is to devise a method of computing the eigenvalues and eigenvectors of A using as much information in $\{u_l\}$ as possible. When one eigenvalue, say λ_{k^*}, is well separated from the others, if l^* is an integer such that $l^*\pi/N$ is the closest to λ_{k^*}, then u_{l^*} will be very close to g_{k^*} subject to a scaling. An estimate of λ_{k^*} is obtained as

$$\tilde{\lambda}|_{l=l^*} = \frac{u_{l^*}^T A u_{l^*}}{u_{l^*}^T u_{l^*}}.$$

Of course, we do not know l^*, but we can compute $\tilde{\lambda}|_l$ for all l, then compare it with $l\pi/N$. When the difference between them is small (or reaches a local minimum), that particular value of l can be saved, and the corresponding u_l and $\tilde{\lambda}|_l$ give rise to an approximate eigenpair of A.

The problem with this approach is that it only works for isolated eigenvalues. If there are two eigenvalues that are close, the value of N may be too small to separate them. Meanwhile, for a multiple eigenvalue, we will only get one eigenvector, which is the component of v_0 on its eigenspace. The method presented below overcomes these difficulties by a subspace method that computes eigenvalues in each well-separated cluster and by adding supplementary vectors when necessary.

Given the vectors $u_0, u_1, \ldots, u_{2N-1}$, we first group them into clusters that are orthogonal with each other and select the most valuable vectors from each cluster.

Recall that related to the polynomial $P_N(z)$, there is a number $\kappa = 1/T_{N-1}((3 - \cos\delta\pi)/(1+\cos\delta\pi))$ such that $|ReP_N(e^{i\theta})| \leq \kappa$ if $|\theta| \leq \delta\pi$ (for $-\pi \leq \theta < \pi$). Apparently, δ is a measure of the narrowness of the peak at $z = 1$ of $P_n(z)$. When δ is chosen (usually $O(1/n)$), we choose N such that κ is smaller than the desired accuracy for eigenvectors. The eigenvalues can be computed as accurately as $O(\kappa^2)$ (but this is also limited by the machine epsilon). Therefore, if we are happy with nine significant digits for the eigenvectors and require full double precision for the eigenvalues, we may choose $\kappa \approx 10^{-9}$. The corresponding N is approximately $8/\delta$.

Let $\epsilon(> \kappa)$ be a small multiple of κ. We retain only the vectors whose two-norms are larger than ϵ. In other words, we find $\{k_j^b, k_j^e\}$ for $j = 1, 2, \ldots, n_c$, such that

$$\begin{aligned} |u_k|_2 &> \epsilon, && \text{if for some } j, \quad k_j^b \leq k \leq k_j^e, \\ |u_k|_2 &\leq \epsilon, && \text{otherwise.} \end{aligned}$$

Thus $\{k_j^b, k_j^e\}$ defines the jth cluster and n_c the total number of clusters. The vectors in different clusters are orthogonal to the accuracy of ϵ.

Inside each cluster, we only retain those vectors whose two-norms are local maxima and not too small (close to ϵ). They are further arranged in decreasing two-norms as

$$W_j = \left[w_1^j, w_2^j, \ldots, w_{p_j}^j\right]$$

for the jth cluster, where p_j is the number of vectors retained in this cluster. The number of all the retained vectors is

$$p = p_1 + p_2 + \cdots + p_{n_c}.$$

If $p = n$, all n eigenpairs can be found based on these chosen vectors. As we will show later, a small symmetric eigenvalue problem with a $p_j \times p_j$ matrix is solved for the jth cluster and the p_j eigenpairs of the matrix A in this cluster are obtained.

When $p < n$, we first need to supplement $n - p$ vectors in order to compute all n eigenpairs. Starting from $n - p$ random vectors, we orthonormalize them with the p retained vectors and obtain

$$W = [w_{p+1}, w_{p+2}, \dots, w_n]$$

satisfying the following conditions:

$$w_k^T w_l^j = 0, \quad \text{for } j = 1, 2, \dots, n_c, \quad l = 1, 2, \dots, p_j, \quad k = p+1, p+2, \dots, n,$$

$$w_k^T w_l = \delta_{kl}, \quad \text{for } k, l = p+1, \quad p+2, \dots, n.$$

Then $W^T A W$ is an $(n - p) \times (n - p)$ symmetric matrix. Its eigenvalue problem is solved, and we obtain the following:

$$W^T A W = Q \begin{pmatrix} \tilde{\mu}_{p+1} & & & \\ & \tilde{\mu}_{p+2} & & \\ & & \ddots & \\ & & & \tilde{\mu}_n \end{pmatrix} Q^T,$$

where Q is an $(n-p) \times (n-p)$ orthogonal matrix. For $\tilde{W} = WQ = [\tilde{w}_{p+1}, \tilde{w}_{p+2}, \dots, \tilde{w}_{p+1}]$, we have

$$\tilde{W}^T A \tilde{W} = \begin{pmatrix} \tilde{\mu}_{p+1} & & & \\ & \tilde{\mu}_{p+2} & & \\ & & \ddots & \\ & & & \tilde{\mu}_n \end{pmatrix}.$$

The $n - p$ vectors of $\tilde{W}$ are grouped into different clusters. For vector $\tilde{w}_k$, we define $\tilde{l}_k$ by

$$\tilde{l}_k = N \tilde{\mu}_k / \pi.$$

If for some j, $k_j^b \le \tilde{l}_k \le k_j^e$, then $\tilde{w}_k$ belongs to the jth cluster. If $\tilde{w}_k$ does not belong to any cluster, then $\{\tilde{\mu}_k, \tilde{w}_k\}$ is an eigenpair of A and no further computation is necessary. This latter case corresponds to the situation where the initial vector has no component in that eigenvector (or the component is so small that we must ignore it). Therefore, each of these $n - p$ vectors is either combined with one of the clusters or left alone as an individual eigenvector. We still denote the number of vectors in the jth cluster by p_j, with the understanding that p_j may have been increased. After an orthonormalization process, we have

$$W_j = [w_1^j, w_2^j, \dots, w_{p_j}^j]$$

such that $(w_k^j)^T w_l^j = \delta_{kl}$. Then the reduced eigenvalue problem for the $p_j \times p_j$ symmetric matrix $W_j^T A W_j$ is solved as a subproblem and we obtain the following:

$$W_j^T A W_j = Q_j \begin{pmatrix} \mu_1^j & & & \\ & \mu_2^j & & \\ & & \ddots & \\ & & & \mu_{p_j}^j \end{pmatrix} Q_j^T,$$

where Q_j is a $p_j \times p_j$ orthogonal matrix. Let

$$\tilde{W}_j = W_j Q_j = [\tilde{w}_1^j, \tilde{w}_2^j, \ldots, \tilde{w}_{p_j}^j].$$

The column vectors of $\tilde{W}_j$ are the eigenvectors of A in the cluster and the corresponding eigenvalues are $\mu_1^j, \mu_2^j, \ldots, \mu_{p_j}^j$.

The method presented in this section selects the most useful vectors from the $2N$ vectors $u_0, u_1, \ldots, u_{2N-1}$, groups them into a number of orthogonal clusters, adds more vectors, if necessary, and reduces to a small symmetric matrix eigenvalue problem in each cluster. We note that the reduced eigenvalue problems in different clusters can be solved simultaneously, therefore, this part of the algorithm is highly parallel. The complete algorithm will be presented in the next section.

7. The algorithm. Consider a real symmetric matrix A. The following steps find the eigenvalues and eigenvectors of A to the desired precision.

1. Scaling and translation. Use any available method, e.g., the Lanczos method, to find the largest and smallest eigenvalues of A approximately, then translate and scale the matrix A, such that the resulting matrix has eigenvalues in the interval $[0, 2\pi)$.

2. Computing the coefficients $\beta_0, \beta_1, \ldots, \beta_{N-1}$. Take $\delta \approx 1/n$, N (power of 2 preferred) is the smallest integer such that

$$\kappa = 1/T_{N-1}((3-\cos\delta\pi)/(1+\cos\delta\pi)) < \text{ desired accuracy for eigenvectors.}$$

The coefficients $\beta_0, \beta_1, \ldots, \beta_{N-1}$ are defined by

$$Q_N(x) = \frac{T_{N-1}((2x+1-\cos(\delta\pi))/(1+\cos(\delta\pi)))}{T_{N-1}((3-\cos(\delta\pi))/(1+\cos(\delta\pi)))} = \sum_{k=0}^{N-1} \beta_k T_k(x),$$

and computed through the recursive formula in §4.

3. Computing the matrix $\cos(A)$. Let $X = A/\pi - I$. The matrices $T_2(X)$, $T_4(X)$, $T_6(X)$, $T_8(X)$, and $T_{10}(X)$ are computed in five matrix multiplications based on $T_{k+l} = 2T_kT_l - T_{k-l}$. Then $\cos(A) = \cos(\pi X)$ is computed in one more matrix multiplication based on the following expansion:

$$\cos(\pi X) \approx c_0 + c_1T_2 + c_2T_4 + \cdots + c_5T_{10} + T_{10}(c_6T_2 + c_7T_4 + \cdots + c_{10}T_{10}),$$

where the coefficients are given in §3.

4. Computing the vectors $v_j = e^{ijA}v_0$. The real part of v_1 is easily obtained from $\cos(A)$. The imaginary part is based on the expansion

$$\begin{aligned} -\sin(A)v_0 &= \sin(\pi X)v_0 \\ &\approx X[s_0 + s_1T_2 + s_2T_4 + \cdots + s_5T_{10} \\ &\qquad + T_{10}(s_6T_2 + s_7T_4 + \cdots + s_{10}T_{10})]v_0. \end{aligned}$$

After v_1 is computed, the other vectors are computed following these steps:

$$\begin{aligned}
&\text{(a)}\quad B_1 = \cos(A),\\
&\qquad v_2 = 2B_1 v_1 - \overline{v}_0;\\
&\text{(b)}\quad B_2 = \cos(2A) = 2B_1^2 - I,\\
&\qquad (v_3, v_4) = 2B_2(v_1, v_2) - \overline{(v1, v0)};\\
&\text{(c)}\quad B_4 = \cos(4A) = 2B_2^2 - I,\\
&\qquad (v_5, v_6, v_7, v_8) = 2B_4(v_1, v_2, v_3, v_4) - \overline{(v_3, v_2, v_1, v_0)};\\
&\text{(d)}\quad \dots\dots\dots\dots
\end{aligned}$$

In practice, the matrices $B_1, B_2, B_4, \dots$ are overwritten and stored in one real symmetric matrix, and all the computed vectors are stored in a complex $n \times N$ matrix $V = [v_0, v_1, v_2, \dots, v_{N-1}]$.

5. Using the Fourier transform. For $\lambda = k\pi/N$, $k = 0, 1, 2, \dots, 2N-1$, the vectors

$$\begin{aligned}
u_k = u(\lambda)|_{\lambda = k\pi/N} &= Re \sum_{j=0}^{N-1} \beta_j v_j e^{-i\pi jk/N}\\
&= \sum_{j=0}^{N-1} \beta_j Re(v_j)\cos(jk\pi/N) + \beta_j Im(v_j)\sin(jk\pi/N)
\end{aligned}$$

are computed through FFT. The matrix V is now overwritten by the real $n \times 2N$ matrix

$$U = [u_0, u_1, \dots, u_{2N-1}].$$

6. Finding the eigensolution. A threshold ϵ is first chosen as a small multiple of κ, the column vectors of U are then grouped into a total of n_c clusters: $\{k_j^b, k_j^e\}$ for $j = 1, 2, \dots, n_c$, such that

$$\begin{aligned}
|u_k|_2 &> \epsilon, && \text{if for some } j, \quad k_j^b \le k \le k_j^e,\\
|u_k|_2 &\le \epsilon, && \text{otherwise.}
\end{aligned}$$

In the jth cluster, p_j vectors are found such that their two-norms reach local maxima. These p_j vectors are ordered with decreasing two-norms and orthonormalized. They are

$$W_j = \left[w_1^j, w_2^j, \dots, w_{p_j}^j\right].$$

$p = p_1 + p_2 + \cdots + p_{n_c}$ is the total number of vectors selected. If $p < n$, $n - p$ random vectors are first generated then orthonormalized with the column vectors of $W_1, W_2, \dots, W_{n_c}$ and we obtain

$$W = [w_{p+1}, w_{p+2}, \dots, w_n].$$

The symmetric matrix eigenvalue problem for the $(n-p) \times (n-p)$ matrix W^TAW is solved:

$$W^TAW = Q \begin{pmatrix} \tilde{\mu}_{p+1} & & & \\ & \tilde{\mu}_{p+2} & & \\ & & \ddots & \\ & & & \tilde{\mu}_n \end{pmatrix} Q^T,$$

where Q is an $(n-p) \times (n-p)$ orthogonal matrix. Let $\tilde{w}_{p+1}, \tilde{w}_{p+2}, \dots, \tilde{w}_{p+1}$ be the $n-p$ column vectors of WQ. For $l = p+1, p+2, \dots, n$, if there exists j such that $k_j^b \le N\tilde{\mu}_l/\pi \le k_j^e$, then the vector $\tilde{w}_l$ is added to the jth cluster. That is,

$$w^j_{p_j+1} = \tilde{w}_l, \qquad p_j := p_j + 1.$$

If $\tilde{w}_l$ cannot be put into any cluster, it is left alone as an eigenvector of A with corresponding eigenvalue $\tilde{\mu}_l$.
Finally, for each cluster W_j, a $p_j \times p_j$ symmetric matrix eigenvalue problem is solved:

$$W_j^TAW_j = Q_j \begin{pmatrix} \mu_1^j & & & \\ & \mu_2^j & & \\ & & \ddots & \\ & & & \mu^j_{p_j} \end{pmatrix} Q_j^T,$$

where Q_j is a $p_j \times p_j$ orthogonal matrix. The column vectors of W_jQ_j are the eigenvectors of A in the jth cluster and the corresponding eigenvalues are $\mu_1^j, \mu_2^j, \dots, \mu^j_{p_j}$.

The method presented above works even if the eigenvalues of A are poorly distributed. However, if many eigenvalues have almost the same value, the work for the supplementary vectors and the resulting clusters is expensive. In most cases, say, for a random matrix, p is close to n, so the operations spent on the supplementary vectors are negligible. Usually, the subproblems in each cluster only involve small matrices. That is, p_j is usually quite small compared with n. These subproblems for supplementary vectors and for the clusters can be solved by the standard EISPACK routines or the same algorithm recursively.

8. Error analysis. For this algorithm to be useful, the computed eigenvalues and eigenvectors must not be sensitive to the initial errors that are present in the matrix A and in computing e^{iA} by an expansion. Since we are using matrix exponentials and repeated squares, we might suspect that our algorithm can be unstable. However, this is not true.

First, we must emphasize that for a symmetric matrix A, e^{iA} is a well-behaved unitary matrix. The usual fear of exponential growth of error does not apply here. The difference between $e^{i(A+E)}$ and e^{iA} is small, when E is a symmetric matrix with small entries. This is related to the fact that the eigenvalues of A and E are real, and the nice properties of sine and cosine are valid here.

Second, while we are computing $v_j = e^{ijA}v_0$ for $j = 1, 2, \dots, N-1$, our results are not based on v_{N-1} and other vectors with large subscripts only. If, at the initial stage, our expansion for e^{iA} gives us an approximation that can be written as $e^{i(A+E)}$, the best we can do for the vector v_j is to obtain $e^{ij(A+E)}v_0$. Indeed, when j is large, the difference between $e^{ijA}v_0$ and $e^{ij(A+E)}v_0$ is significant. However, our computation is based on

$$u_k = Re \sum_{j=0}^{N-1} \beta_j e^{-ijk\pi/N} v_j$$

for $k = 0, 1, \ldots, 2N-1$. If $e^{ij(A+E)}v_0$ is plugged into the above equation for v_j, the computed vectors $\{u_k\}$ simply correspond to the matrix $A+E$, and are very useful when E can be kept small.

The expansion designed for e^{iA} in §3 gives rise to an accurate and reliable scheme for its numerical computation. The computed error for each entry of the matrix is limited by a small multiple of machine epsilon ϵ_0 for double precision (2^{-52} here), which is almost the best we can have. Of course, since we use approximate expansions for both $\cos(A)$ and $\sin(A)$, it is quite impossible to write these two computed matrices in terms of one small symmetric matrix E as $\cos(A+E)$ and $\sin(A+E)$. Rather, we must introduce two small matrices E_1 and E_2 for $\cos(A)$ and $\sin(A)$, respectively. However, the computed imaginary part of v_1, that is, $\sin(A+E_2)v_0$ can be written as $\sin(A+E_1)(v_0+v_e)$, where v_e is a vector with $O(\epsilon_0)$ elements. From the recursive formula, we conclude that for vectors v_j, we obtain

$$\begin{aligned}\hat{v}_j &= \cos(j(A+E_1))v_0 + i\sin(j(A+E_1))(v_0+v_e)\\ &= e^{ij(A+E_1)}v_0 + i\sin(j(A+E_1))v_e.\end{aligned}$$

The computed values for u_k are

$$\begin{aligned}\hat{u}_k &= Re \sum_{j=0}^{N-1} \beta_j e^{-ijk\pi/N} e^{ij(A+E_1)}v_0 + \sum_{j=0}^{N-1} \beta_j \sin(jk\pi/N)\sin(j(A+E_1))v_e\\ &= Re \sum_{j=0}^{N-1} \beta_j e^{-ijk\pi/N} e^{ij(A+E_1)}v_0 + \hat{u}_e.\end{aligned}$$

The first part corresponds to the exact u_k for a perturbed matrix $A+E_1$. The second part $\hat{u}_e$ gives rise to a small error because of the inconsistent approximations for $\sin(A)$ and $\cos(A)$. When v_e is expanded in terms of the eigenvectors of $A+E_1$, say

$$v_e = \hat{\alpha}_1\hat{g}_1 + \hat{\alpha}_n\hat{g}_n + \cdots + \hat{\alpha}_n\hat{g}_n,$$

we see that

$$\hat{u}_e = \sum_{k=1}^{n} \hat{\alpha}_k \sum_{j=0}^{N-1} \beta_j \sin(jk\pi/N)\sin(j\hat{\lambda}_k)\hat{g}_k.$$

The absolute value of the coefficient of $\hat{g}_k$

$$\left| \hat{\alpha}_k \sum_{j=0}^{N-1} \beta_j \sin(jk\pi/N)\sin(j\hat{\lambda}_k) \right| \le |\hat{\alpha}_k| \sum_{j=0}^{N-1} \beta_j = |\hat{\alpha}_k|.$$

Therefore, the two-norm of $\hat{u}_e$ cannot exceed that of v_e, whose entries are on the order of machine epsilon.

The selection of vectors from the set $\{u_k\}$ is based on their two-norms. Those vectors with small norms are ignored. Therefore, the errors $\hat{u}_e$ will not have much impact for our final result and the scheme based on them is stable and reliable.

9. Numerical test. We have tested our algorithm for many matrices. The computed eigenvalues and eigenvectors are compared with the result by EISPACK. The accuracy of the eigenvalues is full double precision, while we limit our accuracy for eigenvectors to nine digits. All the results we computed are found to be correct when compared with those by EISPACK.

The parameter δ is usually taken as $1/n$ and $N = 8/\delta \approx 8n$ is the number of vectors (complex) that we must compute in order to guarantee the nine-digit accuracy for eigenvectors. Unfortunately, δ cannot be taken much larger to reduce the memory space, because we might end up with a situation where the vectors $\{u_k\}$ cannot be separated into different clusters (thus the subproblem is the same size as the original problem).

For random matrices, the eigenvalues are usually well distributed. But some close eigenvalue clusters also exist. Meanwhile, supplementary vectors are usually necessary. However, in all our testing of random matrices, the number of supplementary vectors is significantly less than n and the resulting clusters usually only have a few eigenvalues. For $n = 128$, the size of all the subproblems (for supplementary vectors and for clusters) seldom exceeds 10. Therefore, at least for our choice of δ and N, the time spent on these subproblems is usually not a significant part of the total time. Of course, the sizes of these subproblems depend on the distribution of eigenvalues. If the eigenvalues are extremely clustered, our method can face a difficulty. For simplicity, the subproblems are solved by the EISPACK routines for real and symmetric matrices.

The most time-consuming part of the algorithm is the computation of v_j for $j = 1, 2, \ldots, N-1$. On sequential computers, v_j can be obtained directly by the recursive formula involving $\cos(A)$. Each complex vector v_j requires a multiplication of the real symmetric matrix $\cos(A)$ and the complex vector v_{j-1}. If $N = 8n$, the total number of operations (for both summation and multiplication of real numbers) is approximately $32n^3$. The QR method for the symmetric matrix requires approximately $9n^3$ flops [11] when eigenvectors are also desired. Therefore, this part is already three to four times slower. If we rearrange the computation of v_j as presented in §§5 or 7, we need $\log_2 N - 1$ multiplications of two real symmetric matrices. If a straightforward method is used for matrix multiplications, and $N = 8n$, we need $(\log_2 n+2)n^3$ more operations. Of course, the computation of v_j can now be used in multiplications of real symmetric $n \times n$ matrices with complex matrices of order $n \times 1, n \times 2, n \times 4, \ldots, n \times N/2$. On parallel computers, the efficiency of BLAS3 routines can justify the use of the extra multiplications. A more practical aspect of its advantage over parallel computers will be reported in a separate paper. For the moment, on sequential computers for $n = 128$, the extra work of nine multiplications of real symmetric matrices makes this part of the algorithm four to five times slower than the EISPACK routine.

The computation of $\cos(A)$ requires six multiplications of real symmetric matrices. The total number of operations is $6n^3$. We do not compute $\sin(A)$, since only $\sin(A)v_0$ is needed. The other part of the algorithm is relatively efficient. Although $2N$ could be quite a large number, the step for computing u_k is still efficient, because of the FFT algorithm. The selection and refinement process usually does not involve large subproblems, and it is quite efficient.

Following is a typical example. We consider the matrix

$$A = (\sin(3000(i + j - 2)^2)).$$

For $n = 128$, its eigenvalues are shown in Fig. 1, where each vertical line represents an eigenvalue. We can see that some of the eigenvalues are very close. In our compu-

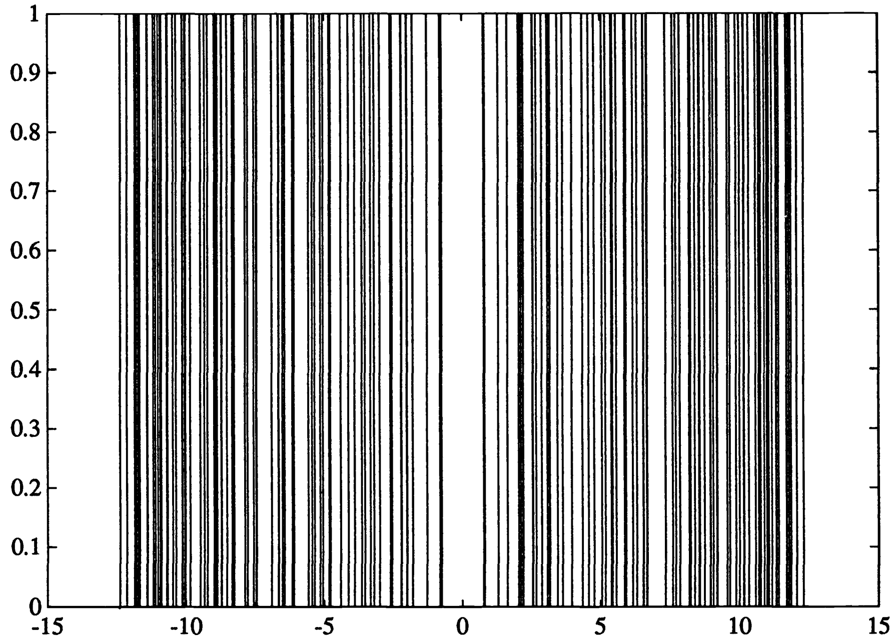


FIG. 1. *Eigenvalues of* $(\sin(3000(i+j-1)^2))$ *for* $n = 128$. (*Each eigenvalue is represented by a vertical line.*)

tation, the parameters are chosen as $\delta = 1/n$, $N = 8n$. With a random initial vector, 11 supplementary vectors are necessary. The result is a total of 59 clusters, classified according to their size (the number of eigenvalues in the cluster) and listed in Table 2. All the eigenvalues are accurate with at least 14 significant digits and the corresponding eigenvectors with nine digits when compared with the results by EISPACK routines. The accuracy of the eigenvectors is controlled in step 2 of the algorithm. Here, we have chosen N and δ such that $\kappa < 10^{-9}$. Therefore, nine accurate digits for the eigenvectors are expected. Let $Q = [q_1, q_2, \dots, q_n]$ be the numerically computed eigenvectors, and $\lambda_1, \lambda_2, \dots, \lambda_n$ be the computed eigenvalues. We have also calculated the following two matrices:

$$R_1 = Q^T Q - I, \qquad R_2 = Q^T A Q - \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}.$$

The entries of these two matrices are all approximately 10^{-9}. Their Frobenius norms (square root of the summation of all the entries squared) are obtained:

$$\frac{1}{n}|R_1|_F = 8.171 \times 10^{-9}, \qquad \frac{1}{n}|R_2|_F = 5.678 \times 10^{-9}.$$

Similar results are obtained for various other testing matrices.

TABLE 2
Eigenvalues in clusters.

Cluster size	Number of clusters
1	24
2	19
3	8
4	4
5	2
6	1
7	0
8	0
9	0
10	1

10. Conclusion. We have shown that for a symmetric $n \times n$ matrix A, its eigenvalues and eigenvectors can be found through a number of matrix multiplications. The number of such multiplications is $O(\log_2 n)$.

The principle of this method is based on the fact that matrix multiplication can be implemented more efficiently than matrix–vector or vector–vector operations. The advantage of BLAS3 operations is expected to be more important in future supercomputers with more processors. Our algorithm starts with six symmetric matrix multiplications to compute $\cos(A)$, then computes intermediate vectors in roughly $\log_2 n$ steps, each involving two matrix multiplications. Although the total number of operations used in our algorithm is much larger than the standard method based on the reduction to tridiagonal matrices, we expect it to be useful on parallel computers in the future.

In theory, the $O(n^3)$ operations required in the straightforward method for matrix multiplication can be reduced significantly. Strassen [25] first discovered a method that requires $O(n^{2.807})$ operations. Later, Pan [19] developed a method that only requires $O(n^{2.496})$ operations. Recently, an $O(n^{2.376})$ method was developed by Coppersmith and Winograd [5]. These methods with lower asymptotic powers are of significant theoretical interest. A number of important problems, such as solving a set of linear systems of equations and inversion of a matrix, can be reduced to matrix multiplications [1]. Our study here links the symmetric eigenvalue problem to the matrix multiplication. An asymptotic efficient algorithm for matrix multiplications can thus be used, which results in an $O(n^{2.376} \log_2 n)$ algorithm for the symmetric eigenvalue problem.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[2] D. BAILEY, *Extra high speed matrix multiplications on the* CRAY-2, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 603–607.

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

[4] R. P. Brent and F. T. Luk, *The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 69–84.

[5] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Proc. 19th Ann. ACM Symp. Theory Comput., 1987, pp. 1–6.

[6] J. Cuppen, *A divide and conquer method for symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195.

[7] J. Dongarra, S. Hammarling, and D. Sorensen, *Block reduction of matrices to condensed form for eigenvalue computations*, Tech. Rep. TM-99, Mathematics and Computer Science Div., Argonne National Laboratory, Argonne, IL, 1987.

[8] J. Dongarra, L. Kaufman, and S. Hammarling, *Squeezing the most out of eigenvalue solvers on high-performance computers*, Linear Algebra Appl., 77 (1986), pp. 113–136.

[9] J. Dongarra and D. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139–s154.

[10] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh, *Parallel algorithms for dense linear algebra computations*, SIAM Rev., 32 (1990), pp. 54–135.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.

[12] W. Harrod, *A block scheme for reduction to condensed form*, Tech. Rep., CSRD Rep. 696, Center for Supercomputing Research and Development, Univ. of Illinois, Argonne, IL, 1988.

[13] H.-M. Huang, *A parallel algorithm for symmetric tridiagonal eigenvalue problems*, CAC Document No. 109, Center for Advanced Computation, Univ. of Illinois at Urbana-Champaign, IL, Feb. 1974.

[14] C. G. J. Jacobi, *Uber ein Leichtes Verfahren Die in der Theorie der Sacularstroungen Vorkommendern Gleichungen Numerisch Aufzulosen*, Crelle's J., 30 (1846), pp. 51–94.

[15] D. Kuck and A. Sameh, *Parallel computation of eigenvalues of real matrices*, IFIP Congress 1971, 2 (1972), pp. 1266–1272.

[16] C. Lanczos, *An iterative method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, Sect. B, 45 (1950), pp. 225–280.

[17] S.-S. Lo, B. Philippe, and A. Sameh, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s155–s165.

[18] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice–Hall, Englewood Cliffs, NJ, 1980.

[19] V. Ya. Pan, *How can we speed up matrix multiplication*? SIAM Rev., 26 (1984), pp. 393–415.

[20] M. S. Paterson and L. J. Stockmeyer, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM J. Comput., 2 (1973), pp. 60–66.

[21] T. J. Rivlin, *Chebyshev Polynomials: From Approximation Theory to Algebraic and Number Theory*, 2nd ed., John Wiley, New York, 1990.

[22] A. Sameh, *On Jacobi and Jacobi-like algorithms for parallel computers*, Math. Comp., 25 (1971), pp. 579–590.

[23] R. Schreiber and C. Van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53–57.

[24] B. T. Smith, J. M. Boyle, J. Dongarra, B. Garbov, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines:* EISPACK *Guide*, 2nd ed., Springer-Verlag, Berlin, New York, 1976.

[25] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[26] C. F. Van Loan, *A note on the evaluation of matrix polynomials*, IEEE Trans. Automat. Control, AC-24 (1978), pp. 320–321.

[27] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, New York, 1965.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 137–158, January 1993

AN IMPLEMENTATION OF THE LOOK-AHEAD LANCZOS ALGORITHM FOR NON-HERMITIAN MATRICES*

ROLAND W. FREUND[†][‡], MARTIN H. GUTKNECHT[§], AND NOËL M. NACHTIGAL[†]

Abstract. The nonsymmetric Lanczos method can be used to compute eigenvalues of large sparse non-Hermitian matrices or to solve large sparse non-Hermitian linear systems. However, the original Lanczos algorithm is susceptible to possible breakdowns and potential instabilities. An implementation of a look-ahead version of the Lanczos algorithm is presented that, except for the very special situation of an incurable breakdown, overcomes these problems by skipping over those steps in which a breakdown or near-breakdown would occur in the standard process. The proposed algorithm can handle look-ahead steps of any length and requires the same number of matrix–vector products and inner products as the standard Lanczos process without look-ahead.

Key words. Lanczos method, orthogonal polynomials, look-ahead steps, eigenvalue problems, iterative methods, non-Hermitian matrices, sparse linear systems

AMS(MOS) subject classifications. 65F15, 65F10

1. Introduction.

In 1950, Lanczos [20] proposed a method for successive reduction of a given, in general non-Hermitian, $N \times N$ matrix A to tridiagonal form. More precisely, the Lanczos procedure generates a sequence $H^{(n)}$, $n = 1, 2, \ldots$, of $n \times n$ tridiagonal matrices which, in a certain sense, approximate A. Furthermore, in exact arithmetic and if no breakdown occurs, the Lanczos method terminates after at most L $(\leq N)$ steps with $H^{(L)}$ a tridiagonal matrix that represents the restriction of A or A^T to an A-invariant or A^T-invariant subspace of $\mathbb{C}^N$, respectively. In particular, all eigenvalues of $H^{(L)}$ are also eigenvalues of A, and, in addition, the method produces basis vectors for the A-invariant or A^T-invariant subspace found.

In the Lanczos process, the matrix A itself is never modified and appears only in the form of matrix–vector products $A \cdot v$ and $A^T \cdot w$. Because of this feature, the method is especially attractive for sparse matrix computations. Indeed, in practice, the Lanczos process is mostly applied to large sparse matrices A, either for computing eigenvalues of A or, in the form of the closely related biconjugate gradient (BCG) algorithm [21], for solving linear systems $Ax = b$. For large A, the finite termination property is of no practical importance and the Lanczos method is used as a purely iterative procedure. Typically, the spectrum of $H^{(n)}$ offers good approximations to some of the eigenvalues of A after already relatively few iterations, i.e., for $n \ll N$. Similarly, BCG, especially if used in conjunction with preconditioning, often converges in relatively few iterations to the solution of $Ax = b$.

Unfortunately, in the standard nonsymmetric Lanczos method a breakdown, more precisely, division by 0, may occur before an invariant subspace is found. In finite-precision arithmetic, such exact breakdowns are very unlikely; however, near-breakdowns may occur, which lead to numerical instabilities in subsequent iterations. The

*Received by the editors April 8, 1991; accepted for publication (in revised form) January 8, 1992.

[†]Research Institute for Advanced Computer Science, Mail Stop T041–5, NASA Ames Research Center, Moffett Field, California 94035. The work of these authors was supported in part by the Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration and the Universities Space Research Association.

[‡]Present Address, AT&T Bell Laboratories, Room 2C-420, 600 Mountain Ave., Murray Hill, New Jersey 07974–0636.

[§]Interdisciplinary Project Center for Supercomputing, ETH Zürich, ETH-Zentrum, CH–8092 Zürich, Switzerland.

possibility of breakdowns has brought the nonsymmetric Lanczos process into discredit and has certainly prevented many people from using the algorithm on non-Hermitian matrices. The symmetric Lanczos process for Hermitian matrices A is a special case of the general procedure in which the occurrence of breakdowns can be excluded.

On the other hand, it is possible to modify the Lanczos process so that it skips over those iterations in which an exact breakdown would occur in the standard method. The related modified recurrences for formally orthogonal polynomials were mentioned by Gragg [14, pp. 222–223] and by Draux [7]; also, in the context of the partial realization problem, by Kung [19, Chap. IV] and Gragg and Lindquist [15]. 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 [16], [17]. Clearly, in finite-precision arithmetic, a viable modified Lanczos process also needs to skip over near-breakdowns. Taylor [27] and Parlett, Taylor, and Liu [25], with their look-ahead Lanczos algorithm, were the first to propose such a practical procedure. However, in [27] and [25], the details of an actual implementation are worked out only for look-ahead steps of length 2. We will use the term *look-ahead Lanczos method* in a broader sense to denote extensions of the standard Lanczos process which skip over breakdowns and near-breakdowns. Finally, note that, in addition to [16] and [17], there are several other recent papers dealing with various aspects of look-ahead Lanczos methods (see [1], [2], [4], [5], [9], [13], [18], and [23]).

The main purpose of this paper is to present a robust implementation of the look-ahead Lanczos method for general complex non-Hermitian matrices. Our intention is to develop an algorithm that can be used as a black box. In particular, the code can handle look-ahead steps of any length and is not restricted to steps of length 2. On many modern computer architectures, the computation of inner products of long vectors is a bottleneck. Therefore, one of our objectives is to minimize the number of inner products in our implementation of the look-ahead Lanczos method. The proposed algorithm requires the same number of inner products as the classical Lanczos process, as opposed to the look-ahead algorithm described in [27] and [25], which always requires additional inner products. In particular, our implementation differs from the one in [27] and [25] even for look-ahead steps of length 2.

The outline of the paper is as follows. In §2, we recall the standard nonsymmetric Lanczos method and its close relationship with orthogonal polynomials. Using this connection, we then describe the basic idea of the look-ahead versions of the Lanczos process. In §3, we present a sketch of our implementation of the algorithm with look-ahead and some of its basic properties. In §4, we discuss in more detail issues related to the look-ahead feature of the algorithm, while in §5 we are concerned with issues related to the implementation of the algorithm. Finally, in §6, we report a few numerical experiments with the algorithm, for both eigenvalue problems and linear systems, and in §7, we make some concluding remarks.

We remark that an extended version of the present paper is available as a technical report [11]. In particular, details omitted here can be found therein. Furthermore, the look-ahead Lanczos process can be used to compute approximate solutions to $Ax = b$, which are defined by a quasi-minimal residual (QMR) property. The resulting QMR algorithm is described in detail in [12] and [13].

For notation, we will adhere to the Householder conventions with only a few exceptions, which we will note. Throughout the paper, all vectors and matrices can be assumed to be complex. As usual, $M^T = [\mu_{ji}]$ and $M^H = [\overline{\mu}_{ji}]$ denote the transpose and the conjugate transpose, respectively, of the matrix $M = [\mu_{ij}]$. The largest and smallest singular values of M are denoted by $\sigma_{\max}(M)$ and $\sigma_{\min}(M)$, respectively. The vector norm $\|x\| = \sqrt{x^H x}$ is always the Euclidean norm and $\|M\| = \sigma_{\max}(M)$ denotes the corresponding matrix norm. The notation

$$K_n(c, B) := \operatorname{span}\{c, Bc, \ldots, B^{n-1}c\}$$

is used for the nth Krylov subspace of $\mathbb{C}^N$ generated by $c \in \mathbb{C}^N$ and the $N \times N$ matrix B.

$$\mathcal{P}_n := \{\Psi(\lambda) \equiv \gamma_0 + \gamma_1\lambda + \cdots + \gamma_n\lambda^n \mid \gamma_0, \gamma_1, \ldots, \gamma_n \in \mathbb{C}\}$$

denotes the set of all complex polynomials of degree at most n. Furthermore, A is always assumed to be a possibly complex and in general non-Hermitian $N \times N$ matrix.

Finally, we note that in our formulation of the nonsymmetric Lanczos algorithm and its look-ahead variant, we use A^T rather than A^H. This was a deliberate choice in order to avoid complex conjugation of the scalars in the recurrences; the algorithms can be formulated equally well in either terms (cf. (2.18)).

2. Background.

In this section, we briefly recall the classical nonsymmetric Lanczos method [20] and its close relationship with *formally orthogonal polynomials* (FOPs). Using this connection, we then describe the basic idea of the look-ahead Lanczos algorithm.

Given two nonzero starting vectors $v_1 \in \mathbb{C}^N$ and $w_1 \in \mathbb{C}^N$, the standard nonsymmetric Lanczos method generates two sequences of vectors $\{v_n\}_{n=1}^L$ and $\{w_n\}_{n=1}^L$ such that, for $n = 1, \ldots, L$,

$$\begin{aligned} \operatorname{span}\{v_1, v_2, \ldots, v_n\} &= K_n(v_1, A), \\ \operatorname{span}\{w_1, w_2, \ldots, w_n\} &= K_n(w_1, A^T), \end{aligned} \tag{2.1}$$

and

$$w_i^T v_j = \left\{ \begin{array}{ll} \delta_i \neq 0 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{array} \right\} \quad \text{for all} \quad i, j = 1, \ldots, n. \tag{2.2}$$

The actual construction of the vectors v_n and w_n is based on the three-term recurrences

$$\begin{aligned} v_{n+1} &= Av_n - \alpha_n v_n - \beta_n v_{n-1}, \\ w_{n+1} &= A^T w_n - \alpha_n w_n - \beta_n w_{n-1}, \end{aligned} \tag{2.3}$$

where

$$\alpha_n = \frac{w_n^T A v_n}{\delta_n} \quad \text{and} \quad \beta_n = \frac{w_{n-1}^T A v_n}{\delta_{n-1}} = \frac{\delta_n}{\delta_{n-1}}$$

are chosen to enforce (2.2). For $n = 1$, we set $\beta_1 = 0$ and $v_0 = w_0 = 0$ in (2.3). Letting

$$V^{(n)} := [v_1\ v_2\ \cdots\ v_n] \quad \text{and} \quad W^{(n)} := [w_1\ w_2\ \cdots\ w_n] \tag{2.4}$$

denote the matrices whose columns are the first n of the vectors v_j and w_j, respectively, and letting

$$H^{(n)} := \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ 1 & \alpha_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_n \\ 0 & \cdots & 0 & 1 & \alpha_n \end{bmatrix}$$

denote the tridiagonal matrix containing the recurrence coefficients, we can rewrite (2.3) as

$$\begin{aligned} AV^{(n)} &= V^{(n)}H^{(n)} + [0 \ \cdots \ 0 \ \ v_{n+1}], \\ A^TW^{(n)} &= W^{(n)}H^{(n)} + [0 \ \cdots \ 0 \ \ w_{n+1}]. \end{aligned} \tag{2.5}$$

Moreover, the biorthogonality condition (2.2) reads as

$$(W^{(n)})^TV^{(n)} = D^{(n)} := \operatorname{diag}(\delta_1, \delta_2, \ldots, \delta_n). \tag{2.6}$$

Let L be the largest integer such that there exist vectors v_n and w_n, $n = 1, \ldots, L$, satisfying (2.1) and (2.2). Note that $L \leq N$ and that, in view of (2.3), L is the smallest integer such that

$$w_{L+1}^Tv_{L+1} = 0. \tag{2.7}$$

Moreover, let

$$L_r = L_r(v_1, A) := \dim K_N(v_1, A) \quad \text{and} \quad L_l = L_l(w_1, A^T) := \dim K_N(w_1, A^T)$$

denote the *grade* of v_1 with respect to A and the *grade* of w_1 with respect to A^T, respectively (cf. [29, p. 37]). There are two essentially different cases for fulfilling the termination condition (2.7). The first case, referred to as *regular termination*, occurs when $v_{L+1} = 0$ or $w_{L+1} = 0$. If $v_{L+1} = 0$, then $L = L_r$ and the *right* Lanczos vectors $v_1, \ldots, v_{L_r}$ span the A-invariant subspace $K_{L_r}(v_1, A)$. Similarly, if $w_{L+1} = 0$, then $L = L_l$ and the *left* Lanczos vectors $w_1, \ldots, w_{L_l}$ span the A^T-invariant subspace $K_{L_l}(w_1, A^T)$. Unfortunately, it can also happen that the termination condition (2.7) is satisfied with $v_{L+1} \neq 0$ and $w_{L+1} \neq 0$. This second case is referred to as *serious breakdown* [29, p. 389]. Note that, in this case,

$$L < L_\star := \min\{L_l, L_r\},$$

and the Lanczos vectors span neither an A-invariant nor an A^T-invariant subspace of $\mathbb{C}^N$.

It is the possibility of serious breakdowns, or, in finite-precision arithmetic, of *near-breakdowns*, i.e.,

$$w_{n+1}^Tv_{n+1} \approx 0, \quad \text{but} \quad w_{n+1} \not\approx 0 \quad \text{and} \quad v_{n+1} \not\approx 0,$$

that has brought the classical nonsymmetric Lanczos algorithm into discredit. However, by means of a look-ahead procedure, it is possible to leap (except in the very special case of an incurable breakdown [27]) over those iterations in which the standard algorithm would break down. Next, using the intimate connection between the Lanczos process and FOPs, we describe the basic idea of the look-ahead Lanczos algorithm.

First, note that

$$\begin{aligned} K_n(v_1, A) &= \{\Psi(A)v_1 \mid \Psi \in \mathcal{P}_{n-1}\}, \\ K_n(w_1, A^T) &= \{\Psi(A^T)w_1 \mid \Psi \in \mathcal{P}_{n-1}\}. \end{aligned} \tag{2.8}$$

In particular, in view of (2.3), for $n = 1, \ldots, L$,

$$v_n = \Psi_{n-1}(A)v_1 \quad \text{and} \quad w_n = \Psi_{n-1}(A^T)w_1, \tag{2.9}$$

where $\Psi_{n-1} \in \mathcal{P}_{n-1}$ is a uniquely defined monic polynomial. Then, introducing the formal inner product

$$(2.10)\qquad (\Phi, \Psi) := \left(\Phi(A^T)w_1\right)^T (\Psi(A)v_1) = w_1^T \Phi(A)\Psi(A)v_1,$$

and using (2.1), (2.8), and (2.9), we can rewrite the biorthogonality condition (2.2) in terms of polynomials:

$$(2.11)\qquad (\Psi_{n-1}, \Psi) = 0 \quad \text{for all} \quad \Psi \in \mathcal{P}_{n-2}$$

and

$$(2.12)\qquad (\Psi_{n-1}, \Psi_{n-1}) \neq 0.$$

Note that, except for the Hermitian case, i.e., $A = A^H$ and $w_1 = \overline{v}_1$, the formal inner product (2.10) is indefinite. Therefore, in the general case, there exist polynomials $\Psi \neq 0$ with "length" $(\Psi, \Psi) = 0$ or even $(\Psi, \Psi) < 0$.

A polynomial $\Psi_{n-1} \in \mathcal{P}_{n-1}$ of exact degree $n-1$ that fulfills (2.11) is called a FOP (with respect to the formal inner product (2.10)); we refer the reader to, e.g., [3], [7], and [16]. Note that the condition (2.11) is empty for $n = 1$, and hence any $\Psi_0 = \gamma_0 \neq 0$ is a FOP of degree 0. From (2.11),

$$\Psi_{n-1}(\lambda) \equiv \gamma_0 + \gamma_1\lambda + \cdots + \gamma_{n-1}\lambda^{n-1}, \quad \text{where} \quad \gamma_{n-1} \neq 0,$$

is a FOP of degree $n-1$ if and only if its coefficients $\gamma_0, \ldots, \gamma_{n-1}$, are a nontrivial solution of the linear system

$$(2.13)\qquad \begin{bmatrix} \mu_0 & \mu_1 & \mu_2 & \cdots & \mu_{n-2} \\ \mu_1 & \iddots & & \iddots & \vdots \\ \mu_2 & & \iddots & & \vdots \\ \vdots & \iddots & & & \mu_{2n-5} \\ \mu_{n-2} & \cdots & \cdots & \mu_{2n-5} & \mu_{2n-4} \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{n-2} \end{bmatrix} = -\gamma_{n-1} \begin{bmatrix} \mu_{n-1} \\ \mu_n \\ \mu_{n+1} \\ \vdots \\ \mu_{2n-3} \end{bmatrix}.$$

Here

$$\mu_j := w_1^T A^j v_1 = (1, \lambda^j), \qquad j = 0, 1, \ldots,$$

are the moments associated with (2.10). A FOP Ψ_{n-1} is called *regular* if it is uniquely determined by (2.11) up to a scalar, and it is said to be *singular* otherwise. We remark that FOPs of degree 0 are always regular. In particular, a regular FOP is unique if it is required to be monic. Moreover, singular FOPs occur if and only if the corresponding linear system (2.13) has a singular coefficient matrix, but is consistent for some $\gamma_{n-1} \neq 0$. If (2.13) is inconsistent when $\gamma_{n-1} \neq 0$, then no FOP Ψ_{n-1} exists. This case is referred to as *deficient*. By relaxing (2.11) slightly, one can define so-called *deficient FOPs* (see [16] for details). Simple examples (see, e.g., [12, §13]) show that the singular and deficient cases do indeed occur. Thus, regular FOPs need not exist for every degree $n-1$. We would like to stress that this phenomenon is due to the indefiniteness of (2.10). For a positive definite inner product $(\cdot,\cdot)$, unique monic formally orthogonal polynomials always exist, up to the degree equal to the grade of v_1 with respect to A. Such a definite inner product is induced in the Hermitian case $A = A^H$ and $w_1 = \overline{v_1}$. In this case, the FOPs are true orthogonal polynomials with respect to a positive weight whose support is a set of points on the real axis (see, e.g., [26]). In addition, they have real coefficients and therefore

$$(\Psi, \Psi) = w_1^T \Psi(A)\Psi(A)v_1 = v_1^H \overline{\Psi}(A^H)\Psi(A)v_1 = \|\Psi(A)v_1\|^2 .$$

Finally, given a regular FOP Ψ_{n-1}, it is easily checked whether a regular FOP of degree n exists. Indeed, using (2.13), one readily obtains the following lemma.

LEMMA 2.1. *Let Ψ_{n-1} be a regular* FOP *(with respect to the formal inner product* (2.10)) *of degree $n-1$. Then, a regular* FOP *of degree n exists if and only if* (2.12) *is satisfied.*

Let us return to the standard nonsymmetric Lanczos process (2.3). Using (2.7), (2.9), (2.10), and Lemma 2.1, we conclude that a serious breakdown occurs if and only if no regular FOP exists for some $L < L_\star$. In this case, the termination index L is the smallest integer L for which there exists no regular FOP of degree L.

On the other hand, there is a maximal subset of indices

$$(2.14) \quad \{n_1, n_2, \ldots, n_J\} \subseteq \{1, 2, \ldots, L_\star\}, \qquad n_1 := 1 < n_2 < \cdots < n_J \le L_\star,$$

such that, for each $j = 1, 2, \ldots, J$, there exists a monic regular FOP $\Psi_{n_j - 1} \in \mathcal{P}_{n_j - 1}$. Note that $n_1 = 1$ since $\Psi_0(\lambda) \equiv 1$ is a monic regular FOP of degree 0. It is well known [7], [15] that three successive regular FOPs $\Psi_{n_{j-1}-1}$, $\Psi_{n_j - 1}$, and $\Psi_{n_{j+1}-1}$ are connected via a three-term recurrence. Consequently, setting, in analogy to (2.9),

$$v_{n_j} = \Psi_{n_j - 1}(A)v_1 \quad \text{and} \quad w_{n_j} = \Psi_{n_j - 1}(A^T)w_1,$$

we obtain two sequences of vectors $\{v_{n_j}\}_{j=1}^J$ and $\{w_{n_j}\}_{j=1}^J$ which can be computed by means of three-term recurrences. These vectors will be called *regular* vectors, since they correspond to regular FOPs. Note that the starting vectors v_1 and w_1 are always regular. The look-ahead Lanczos procedure is an extension of the classical nonsymmetric Lanczos algorithm; in exact arithmetic, it generates the vectors v_{n_j} and w_{n_j}, $j = 1, \ldots, J$. If $n_J = L_\star$ in (2.14), then these vectors can be complemented to a basis for an A-invariant or A^T-invariant subspace of $\mathbb{C}^N$. An incurable breakdown occurs if and only if $n_J < L_\star$ in (2.14). Finally, note that

$$w_{n_j}^T v = w^T v_{n_j} = 0 \quad \text{for all} \quad v \in K_{n_j - 1}(v_1, A), \; w \in K_{n_j - 1}(w_1, A^T), \quad j = 1, \ldots, J.$$

The look-ahead procedure we have sketched so far only skips over exact breakdowns. It yields what is called the *nongeneric* Lanczos algorithm in [16]. Of course, in finite-precision arithmetic, the look-ahead Lanczos algorithm also needs to leap over near-breakdowns. Roughly speaking, a robust implementation should attempt to generate only the "well-defined" regular vectors. In practice, then, one aims to generate two sequences of vectors $\{v_{n_{j_k}}\}_{k=1}^K$ and $\{w_{n_{j_k}}\}_{k=1}^K$ where

$$(2.15) \qquad \{n_{j_k}\}_{k=1}^K \subseteq \{n_j\}_{j=1}^J, \qquad j_1 := 1,$$

is a suitable subset of (2.14). We set $j_1 = 1$, since v_1 and w_1 are always regular. The problem of how to determine the set (2.15) of indices of the "well-defined" regular vectors will be addressed in detail in §4.

In order to obtain complete bases for the subspaces $K_n(v_1, A)$ and $K_n(w_1, A^T)$, we need to add vectors

$$(2.16) \quad \begin{aligned} &v_n \in K_n(v_1, A) \setminus K_{n-1}(v_1, A) \quad \text{and} \quad w_n \in K_n(w_1, A^T) \setminus K_{n-1}(w_1, A^T), \\ &\qquad\qquad n = n_{j_{k-1}} + 1, \ldots, n_{j_k} - 1, \qquad k = 2, 3, \ldots, K, \end{aligned}$$

to the two sequences $\{v_{n_{j_k}}\}_{k=1}^K$ and $\{w_{n_{j_k}}\}_{k=1}^K$, respectively. Clearly, (2.16) guarantees that (2.1) remains valid for the look-ahead Lanczos algorithm. The vectors in (2.16) are called *inner* vectors. Moreover, for each k, the vectors v_n, $n = n_{j_k}, n_{j_k}+1, \ldots, n_{j_{k+1}}-1$, and correspondingly for w_n, are referred to as the kth *block*. The inner vectors of a block built because of an exact breakdown correspond to singular or deficient FOPs, while the inner vectors of a block built because of a near-breakdown correspond to polynomials that in general are combinations of regular, singular, and deficient FOPs. We will refer to both the regular and the inner vectors v_n and w_n generated by the look-ahead variant as right and left Lanczos vectors, in analogy to the terminology of the standard nonsymmetric Lanczos algorithm.

So far, we have not specified how to actually construct the inner vectors. The point is that the inner vectors can be chosen such that the v_n's and w_n's from blocks corresponding to different indices k are still biorthogonal to each other. More precisely, with $V^{(n)}$ and $W^{(n)}$ defined as in (2.4), we have, in analogy to (2.6),

$$(2.17)\qquad (W^{(n)})^T V^{(n)} = D^{(n)}, \quad n = n_{j_l}-1, \quad l = 2,3,\ldots,K.$$

Here, $D^{(n)}$ is now a nonsingular block diagonal matrix with $l-1$ blocks of respective size $(n_{j_{k+1}}-n_{j_k})\times(n_{j_{k+1}}-n_{j_k})$, $k = 1,\ldots,l-1$. Similarly, (2.5) holds, for $n = n_{j_l}-1$, $l = 2,3,\ldots,K$, where $H^{(n)}$ is now a block tridiagonal matrix with diagonal blocks of size $(n_{j_{k+1}}-n_{j_k})\times(n_{j_{k+1}}-n_{j_k})$, $k = 1,\ldots,l-1$ (cf. (3.4)–(3.5)).

There are two fundamentally different approaches for constructing inner vectors with the property (2.17). In both cases, inner vectors are first generated using a simple three-term recurrence. However, in the first approach, each inner vector in a block is then biorthogonalized against the previous block as soon as it is constructed. This variant will be called the *sequential algorithm*. In the second approach, all the inner vectors in a block are first constructed using the three-term recurrence, and then the entire block is biorthogonalized against the previous block and possibly, depending on the size of the current block, against vectors from blocks further back. This variant will be called the *block algorithm*. The sequential algorithm is more suitable for a serial computer, while the block algorithm is more suitable for a parallel computer. In this paper, we describe only the sequential algorithm and its implementation. A sketch of the block algorithm can be found in [11]. Details of an actual implementation and numerical results will be presented elsewhere.

Finally, two more notes. First, the inner product (2.10) could have been defined as

$$(2.18)\qquad (\Phi,\Psi) := \left(\overline{\Phi}(A^H)\overline{w_1}\right)^H (\Psi(A)v_1) = \overline{w_1}^H \Phi(A)\Psi(A)v_1,$$

and the algorithm could be formulated equally well in either terms. Second, in the rest of the paper, we will use the notation $n_k := n_{j_k}$ for the indices of the "well-defined" regular vectors. However, notice that there is no guarantee that the indices n_k generated by the look-ahead Lanczos algorithm in finite-precision arithmetic actually satisfy (2.15).

3. The sequential algorithm. In this section, we start the discussion of the sequential Lanczos algorithm with look-ahead. We present a sketch of the algorithm and its basic properties, then discuss some aspects related to its practical implementation in the next two sections.

First, we introduce some notation. As in the last section, $n = 1,2,\ldots,$ denote the indices of the Lanczos vectors v_n and w_n. The index $k = 1,2,\ldots,$ is used as a counter for the blocks built by the look-ahead algorithm. Moreover, we always use $l = l(n)$ to denote the index of the block which contains the Lanczos vectors v_n and w_n. Recall that by n_k we denote the indices of the computed regular vectors, which are always the first vectors in each block k. Thus, n_l is the index of the last computed regular vector with index $\leq n$. We have $n_1 = 1$. Capital letters with subscript k denote the matrices containing quantities from block k. For example,

$$V_k := \begin{bmatrix} v_{n_k} & v_{n_k+1} & \cdots & v_{n_{k+1}-1} \end{bmatrix}$$

is the matrix whose columns are the Lanczos vectors from a completed block k. Capital letters with superscripts $^{(n)}$ denote matrices containing quantities from steps 1 through n, as in (2.4). With this notation, the matrix form of the sequential algorithm with look-ahead is similar to (2.5)–(2.6)

$$\begin{aligned} AV^{(n)} &= V^{(n)}H^{(n)} + \begin{bmatrix} 0 & \cdots & 0 & v_{n+1} \end{bmatrix}, \\ A^TW^{(n)} &= W^{(n)}H^{(n)} + \begin{bmatrix} 0 & \cdots & 0 & w_{n+1} \end{bmatrix}, \end{aligned} \tag{3.1}$$

and

$$(W^{(n)})^TV^{(n)} = D^{(n)}. \tag{3.2}$$

Here

$$D^{(n)} = \operatorname{diag}(\delta_1, \delta_2, \ldots, \delta_l), \quad \delta_k := W_k^TV_k, \quad k = 1, 2, \ldots, l = l(n), \tag{3.3}$$

is block diagonal, and the blocks $\delta_1, \delta_2, \ldots, \delta_{l-1}$ are nonsingular. If $n = n_{l+1} - 1$, then the lth block, δ_l, in (3.3) is also nonsingular and it is called *complete*. In particular, if δ_l is complete, then $D^{(n)}$ itself is nonsingular, and (3.3) reduces to (2.17). In this case, the next regular vectors $v_{n_{l+1}}$ and $w_{n_{l+1}}$ can be computed and start a new block.

In (3.1),

$$H^{(n)} = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \gamma_2 & \alpha_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_l \\ 0 & \cdots & 0 & \gamma_l & \alpha_l \end{bmatrix} \tag{3.4}$$

is an $n \times n$ block tridiagonal upper Hessenberg matrix with blocks of the form

$$\alpha_k = \begin{bmatrix} * & \cdots & \cdots & \cdots & * \\ 1 & \ddots & & & \vdots \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & * \end{bmatrix}, \qquad \gamma_k = \begin{bmatrix} 0 & \cdots & \cdots & 0 & 1 \\ \vdots & \ddots & & & 0 \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix}, \tag{3.5}$$

while the β_k's are in general full matrices. Note that here we violate the Householder conventions by using small Greek letters to denote quantities which may be matrices. The justification is that in general the algorithm takes regular steps, and hence these quantities are usually scalars. Let $h_k := n_{k+1} - n_k$, $k = 1, 2, \ldots$, be the size of the kth block. For $k < l = l(n)$ the matrices α_k, β_k, and γ_k are of size $h_k \times h_k$, $h_{k-1} \times h_k$, and $h_k \times h_{k-1}$, respectively. In general, however, the lth block need not be complete. Hence, the matrices α_l, β_l, and γ_l corresponding to the current (lth) block are of size $\tilde{h}_l \times \tilde{h}_l$, $h_{l-1} \times \tilde{h}_l$, and $\tilde{h}_l \times h_{l-1}$, respectively, where $\tilde{h}_l := n + 1 - n_l$.

We will assume that the inner vectors in a block are generated using a three-term recursion of the form

$$\begin{aligned} v_{n+1} &= Av_n - \zeta_n v_n - \eta_n v_{n-1}, \\ w_{n+1} &= A^T w_n - \zeta_n w_n - \eta_n w_{n-1}, \end{aligned} \tag{3.6}$$

where ζ_n and η_n are recursion coefficients and $\eta_{n_k} = 0$, $k = 1, 2, \ldots$. One may choose these coefficients so that they remain the same from one block to the next and change only with respect to their index inside the block, $n - n_k$, or one may choose these coefficients so that they change from one block to the next. For instance, one practical choice for the polynomials in (3.6) are suitably scaled and translated Chebyshev polynomials, so that the inner vectors are generated by the Chebyshev iteration [22]. In this case, the translation parameters could be adjusted using spectral information obtained from previous Lanczos steps. We do not necessarily advocate the use of fancy recursions in (3.6). From our experience, the algorithm we propose builds very small blocks, typically of size 2 or 3. Except for p-cyclic matrices (cf. Example 6.3 in §6) or contrived examples, the largest block we observed in test runs with "real-life" matrices was of size 4. It occurred for a matrix arising in oil-reservoir simulations where out of 1500 steps, the algorithm built 2×2 blocks 49 times, 3×3 blocks 7 times, and one 4×4 block (see [13, Ex. 2]). Hence, the recursion in (3.6) is not overly important, and in our experiments, we have used the recursion coefficients $\zeta_n = 1$ and, if $n \neq n_k$, $\eta_n = 1$. On the other hand, for the block version of the algorithm, where larger blocks are built, more attention needs to be paid to the recursion used. As indicated, details of the block algorithm will be presented elsewhere. Finally, one could consider orthogonalizing (in the Euclidean sense) the right, respectively, left, Lanczos vectors within each block. However, for the blocks we have seen built, such an orthogonalization process did not lead to better numerical properties of the algorithm. Therefore, in view of the additional inner products that need to be computed, orthogonalizing within each block is not justified.

In practice, for reasons of stability, one computes scaled versions of the right and left Lanczos vectors, rather than the "monic" vectors v_n and w_n corresponding to monic FOPs. A proven choice (see [25] and [27]) is to scale the Lanczos vectors to have unit length. We denote by $\hat{v}_n$ and $\hat{w}_n$ the scaled versions defined by

$$\hat{v}_n := v_n / \|v_n\| \quad \text{and} \quad \hat{w}_n := w_n / \|w_n\|,$$

and more generally, we will denote by hat ($\hat{}$) quantities containing or depending on the scaled vectors. For example, setting

$$\hat{H}^{(n)} := \operatorname{diag}(\|v_1\|, \|v_2\|, \ldots, \|v_n\|) H^{(n)} \operatorname{diag}(1/\|v_1\|, 1/\|v_2\|, \ldots, 1/\|v_n\|),$$

$$\hat{H}_e^{(n)} := \begin{bmatrix} \hat{H}^{(n)} \\ \rho_{n+1} e_n^T \end{bmatrix}, \quad \rho_{n+1} := \frac{\|v_{n+1}\|}{\|v_n\|}, \quad e_n := [0 \quad \cdots \quad 0 \quad 1]^T \in \mathbb{R}^n,$$

we can rewrite the first relation in (3.1) in terms of scaled vectors as follows:

$$A\hat{V}^{(n)} = \hat{V}^{(n+1)} \hat{H}_e^{(n)}. \tag{3.7}$$

With this note, we now present a sketch of the sequential Lanczos algorithm with look-ahead.

```
ALGORITHM 3.1 (SEQUENTIAL LANCZOS ALGORITHM WITH LOOK-AHEAD).
(0) Choose $\hat{v}_1,\ \hat{w}_1 \in \mathbb{C}^N$ with $\|\hat{v}_1\| = \|\hat{w}_1\| = 1$;
    Set $\hat{V}_1 = \hat{v}_1,\ \hat{W}_1 = \hat{w}_1,\ \hat{\delta}_1 = \hat{W}_1^T \hat{V}_1$;
    Set $n_1 = 1,\ l = 1,\ \hat{v}_0 = \hat{w}_0 = 0,\ \hat{V}_0 = \hat{W}_0 = \emptyset,\ \rho_1 = \xi_1 = 1$;
For $n = 1, 2, \ldots$ :
(1) Decide whether to construct $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ as regular or inner vectors
    and go to (2) or (3), respectively;
(2) (Regular step.) Compute
```

$$\begin{aligned} \tilde{v}_{n+1} &= A\hat{v}_n - \hat{V}_l\ \hat{\delta}_l^{-1}\hat{W}_l^T A\hat{v}_n - \hat{V}_{l-1}\hat{\delta}_{l-1}^{-1}\hat{W}_{l-1}^T A\hat{v}_n, \\ \tilde{w}_{n+1} &= A^T\hat{w}_n - \hat{W}_l\ \hat{\delta}_l^{-T}\hat{V}_l^T A^T\hat{w}_n - \hat{W}_{l-1}\hat{\delta}_{l-1}^{-T}\hat{V}_{l-1}^T A^T\hat{w}_n, \end{aligned} \tag{3.8}$$

```
    set $n_{l+1} = n + 1,\ l = l + 1,\ \hat{V}_l = \hat{W}_l = \emptyset$, and go to (4);
(3) (Inner step.) Compute
```

$$\begin{aligned} \tilde{v}_{n+1} &= A\hat{v}_n - \zeta_n\hat{v}_n - (\eta_n/\rho_n)\,\hat{v}_{n-1} - \hat{V}_{l-1}\hat{\delta}_{l-1}^{-1}\hat{W}_{l-1}^T A\hat{v}_n, \\ \tilde{w}_{n+1} &= A^T\hat{w}_n - \zeta_n\hat{w}_n - (\eta_n/\xi_n)\,\hat{w}_{n-1} - \hat{W}_{l-1}\hat{\delta}_{l-1}^{-T}\hat{V}_{l-1}^T A^T\hat{w}_n; \end{aligned} \tag{3.9}$$

```
(4) Compute $\rho_{n+1} = \|\tilde{v}_{n+1}\|$ and $\xi_{n+1} = \|\tilde{w}_{n+1}\|$;
    If $\rho_{n+1} = 0$ or $\xi_{n+1} = 0$, stop;
    Otherwise, set
```

$$\begin{aligned} &\hat{v}_{n+1} = \tilde{v}_{n+1}/\rho_{n+1}, \quad \hat{w}_{n+1} = \tilde{w}_{n+1}/\xi_{n+1}, \\ &\hat{V}_l = [\,\hat{V}_l \quad \hat{v}_{n+1}\,], \quad \hat{W}_l = [\,\hat{W}_l \quad \hat{w}_{n+1}\,], \quad \hat{\delta}_l = \hat{W}_l^T\hat{V}_l; \end{aligned} \tag{3.10}$$

```
(5) Add the nonzero elements of the nth column to $\hat{H}^{(n)}$
    and set $(\hat{H}_e^{(n)})_{n+1,n} = \rho_{n+1}$.
```

Note that, if $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ are inner vectors, the size of the current incomplete block l is increased by 1; if they are regular vectors, then the lth block is complete and a new block, the $(l+1)$st, is started, with $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ as its first vectors. Finally, we remark that, in view of (3.7), the nonzero elements of the nth column of $\hat{H}^{(n)}$ occur as coefficients in the first recursion of (3.8), respectively, (3.9).

4. Building blocks.

In this section, we discuss the criteria used in step (1) of Algorithm 3.1 to decide whether a pair of Lanczos vectors $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ is built as inner vectors or as regular vectors. We propose three criteria, namely, (4.3), (4.4), and (4.5) below. If all three checks (4.3)–(4.5) are satisfied, then $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ are constructed as regular vectors; otherwise, they are constructed as inner vectors. Let us motivate these three criteria.

First, recall (cf. (3.3)) that for $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ to be built as regular vectors it is necessary that $\hat{\delta}_l$ be nonsingular. Therefore, it is tempting to base the decision "regular versus inner step" solely on checking whether $\hat{\delta}_l$ is close to singular, and to perform a regular step if and only if

$$\sigma_{\min}(\hat{\delta}_{l(n)}) \geq \text{tol}, \tag{4.1}$$

for some suitably chosen tolerance tol. For example, Parlett [23] suggests tol $= \epsilon^{1/4}$ or tol $= \epsilon^{1/3}$, where ϵ denotes the roundoff unit. Then (4.1) would guarantee that complete blocks of computed Lanczos vectors satisfy

$$\sigma_{\min}(\hat{\delta}_k) \geq \text{tol}, \qquad k = 1, 2, \ldots.$$

This, together with (3.3), would imply by [23, Thm. 10.1] that

$$\sigma_{\min}(\hat{V}^{(n)}) \geq \frac{\text{tol}}{\sqrt{n}} \quad \text{and} \quad \sigma_{\min}(\hat{W}^{(n)}) \geq \frac{\text{tol}}{\sqrt{n}}, \quad n = n_k - 1, \quad k = 1, 2, \ldots. \tag{4.2}$$

Since the columns of $\hat{V}^{(n)}$ and $\hat{W}^{(n)}$ are unit vectors, $\sigma_{\min}(\hat{V}^{(n)})$ and $\sigma_{\min}(\hat{W}^{(n)})$ are a measure of the linear independence of these vectors; in particular, (4.2) would ensure that the Lanczos vectors remain linearly independent. However, in the outlined algorithm, the block orthogonality (3.2)–(3.3) is enforced only among two or three successive blocks, and in finite-precision arithmetic, biorthogonality of blocks whose indices are far apart is typically lost. The theorem assumes that (3.2)–(3.3) hold for all indices, and without this, the theorem fails in finite arithmetic. We illustrate this with a simple example.

Example 4.1. In Fig. 4.1, we plot $\sigma_{\min}(\hat{\delta}_{l(n)})$ (dots), $\min_{1\leq k<l(n)}(\sigma_{\min}(\hat{\delta}_k))$ (solid line), and $\sqrt{n}\,\sigma_{\min}(\hat{V}^{(n)})$ (dotted line), as functions of the iteration index $n = 1, 2, \ldots$, for a random 50×50 dense matrix. The theorem predicts that

$$\sqrt{n}\,\sigma_{\min}(\hat{V}^{(n)}) \geq \min_{1\leq k<l(n)}(\sigma_{\min}(\hat{\delta}_k)),$$

which is clearly not the case.

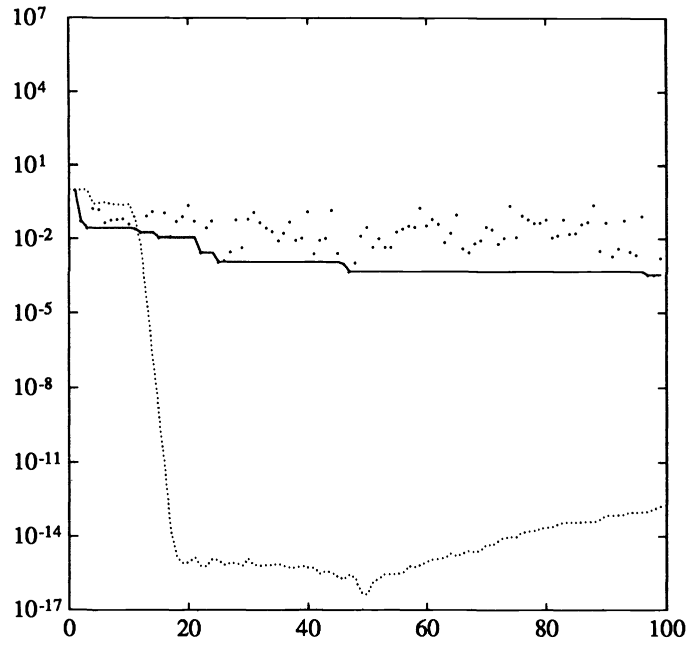


FIG. 4.1. $\sigma_{\min}(\hat{\delta}_{l(n)})$ *(dots),* $\min_{1\leq k<l(n)}(\sigma_{\min}(\hat{\delta}_k))$ *(solid line), and* $\sqrt{n}\,\sigma_{\min}$ $(\hat{V}^{(n)})$ *(dotted line), plotted versus the iteration index n.*

As this simple example shows, the check (4.1) alone does not ensure that the computed Lanczos vectors are sufficiently linearly independent. In particular, if the look-ahead strategy is based only on criterion (4.1), the algorithm may produce within a block Lanczos vectors that are almost linearly dependent. When this happens, the check (4.1) usually fails in all subsequent iterations and thus the algorithm never completes the current block, i.e., it has generated an *artificial* incurable breakdown.

In addition, numerical experience indicates another problem with (4.1): for values of tol that are "reasonably" larger than machine epsilon, the behavior of the algorithm is very sensitive with respect to the actual value of tol. We also illustrate this with an example.

Example 4.2. We applied the Lanczos algorithm to a nonsymmetric matrix A obtained from discretizing a three-dimensional partial differential equation (cf. Example 6.5 in §6). This example was run on a machine with $\epsilon \approx$ 1.3E-29. In the first case, we set tol $= \epsilon^{1/4} \approx$ 6.0E-08, while in the second case, we set tol $= \epsilon^{1/3} \approx$ 2.3E-10. In Fig. 4.2, we plot $\sigma_{\min}(\hat{\delta}_{l(n)})$ versus the iteration index n for the two runs, the dotted line for $\epsilon^{1/4}$ and the solid line for $\epsilon^{1/3}$. In the first case, the algorithm starts building a block that it never closes, and the singular values clearly become smaller and smaller. Yet if tol is only slightly smaller, as in the second case, the algorithm runs to completion, in this case solving the linear system to the desired accuracy, and thus indicating that the block built in the first case was not a true, but an artificial incurable breakdown.

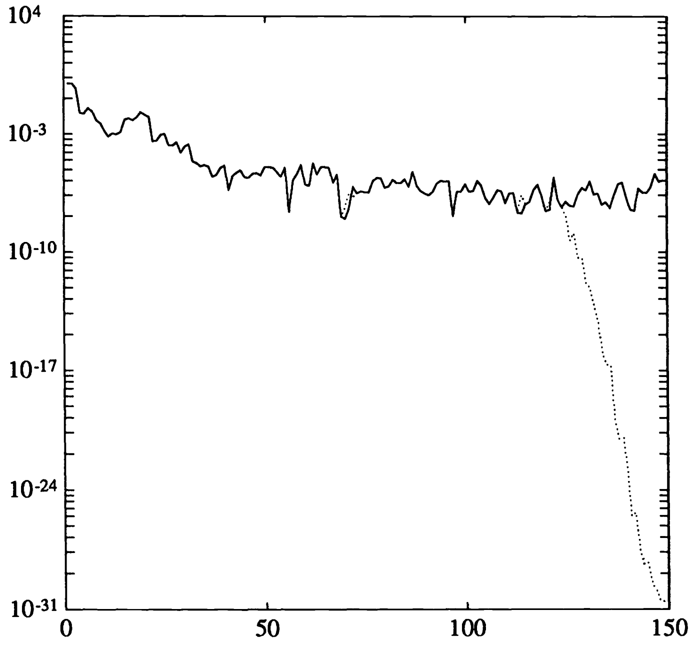


FIG. 4.2. $\epsilon^{1/4}$ *(dotted line) and* $\epsilon^{1/3}$ *(solid line), plotted versus the iteration index* n.

We note that the sensitivity of look-ahead procedures to the choice of tolerances, such as tol in (4.1), was also observed in [5]. However, no remedy for this phenomenon is given in [5]. Furthermore, we remark that the problem of generating almost linearly dependent vectors is not specific to the Lanczos biorthogonalization process. Indeed, similar effects can also occur in true orthogonalization methods (cf. [28]).

Examples 4.1 and 4.2 clearly show that the decision "regular versus inner step" cannot be based on (4.1) alone. Instead, we propose to relax the check (4.1), so that it merely ensures that $\hat{\delta}_{l(n)}$ is numerically nonsingular, and to add the checks (4.4)–(4.5) below, which guarantee that the computed Lanczos vectors remain sufficiently linearly independent. Hence, instead of (4.1), we check for

$$\sigma_{\min}(\hat{\delta}_{l(n)}) \geq \epsilon, \tag{4.3}$$

where ϵ denotes the roundoff unit.

Our numerical experiments have shown that typically the algorithm starts to generate Lanczos vectors which are almost linearly dependent, once a regular vector $\hat{v}_{n+1}$ was computed whose component $A\hat{v}_n \in K_{n+1}(v_1, A)$ is dominated by its component in the previous Krylov space $K_n(v_1, A)$ (and similarly for $\hat{w}_{n+1}$). In order to avoid the construction of such regular vectors, we check the l_1-norm of the coefficients for $\hat{V}_{l-1}$ and $\hat{V}_l$ in (3.8); $\hat{v}_{n+1}$ can be computed as a regular vector only if

$$\sum_{j=n_{l-1}}^{n_l-1} \left|(\hat{\delta}_{l-1}^{-1}\hat{W}_{l-1}^T A\hat{v}_n)_j\right| \leq n(A) \quad \text{and} \quad \sum_{j=n_l}^{n} \left|(\hat{\delta}_l^{-1}\hat{W}_l^T A\hat{v}_n)_j\right| \leq n(A). \tag{4.4}$$

Here $n(A)$ is a factor depending on the norm of A; we will indicate later how this factor is computed. Similarly, we check the l_1-norm of the coefficients for $\hat{W}_{l-1}$ and $\hat{W}_l$ in (3.8); $\hat{w}_{n+1}$ can be computed as a regular vector only if

$$\sum_{j=n_{l-1}}^{n_l-1} \left|(\hat{\delta}_{l-1}^{-T}\hat{V}_{l-1}^T A^T\hat{w}_n)_j\right| \leq n(A) \quad \text{and} \quad \sum_{j=n_l}^{n} \left|(\hat{\delta}_l^{-T}\hat{V}_l^T A^T\hat{w}_n)_j\right| \leq n(A). \tag{4.5}$$

The pair $\hat{v}_{n+1}$ and $\hat{w}_{n+1}$ is built as regular vectors only if the checks (4.3)–(4.5) hold true.

We need to indicate how $n(A)$ is chosen in (4.4)–(4.5). Numerical experience with matrices whose norm is known indicates that setting $n(A) = \|A\|$ is too strict and can result in artificial incurable breakdowns. A better setting seems to be $n(A) = 10 \cdot \|A\|$, but even this is dependent on the matrix. In any case, in practice one does not know $\|A\|$, and there is also the issue of a maximal block size, determined by limits on available storage. To solve the problems of estimating the norms and a suitable factor $n(A)$, as well as cope with limited storage and yet allow the algorithm to proceed as far as possible, we propose the following procedure. Suppose we are given an initial value for $n(A)$, based either on an estimate from the user (for example, $n(A)$ from a previous run with the matrix A), or by setting

$$n(A) = \max\left\{\|A\hat{v}_1\|, \|A^T\hat{w}_1\|\right\}.$$

Note that here A denotes the matrix actually used in generating the Lanczos vectors, thus including the case when we are solving a preconditioned linear system. We then update $n(A)$ dynamically, as follows. In each block, whenever an inner vector is built because one of the checks (4.4)–(4.5) is not satisfied, the algorithm keeps track of the size of the terms that have caused one or more of (4.4)–(4.5) to be false. If the block closes naturally, then this information is not needed. If, however, the algorithm is about to run out of storage, then $n(A)$ is replaced with the smallest value which has caused an inner vector to be built. The updated value of $n(A)$ is guaranteed to pass the checks (4.4)–(4.5) at least once, and hence the block is guaranteed to close. This also frees up the storage that was used by the previous block, thus ensuring that the algorithm can proceed.

5. Implementation details. We now turn to a few implementation details. In particular, we wish to show how one can implement the sequential algorithm with the same number of inner products per step as the classical Lanczos algorithm. For a regular step, one needs to compute $\hat{\delta}_l$, $\hat{W}_l^T A\hat{v}_n$, and $\hat{W}_{l-1}^T A\hat{v}_n$ in (3.8). For an inner step, one needs to compute $\hat{W}_{l-1}^T A\hat{v}_n$ in (3.9) and to update $\hat{\delta}_l$ in (3.10). We will show that for a block of size h_l, only $2h_l$ inner products are required: $2h_l - 1$ will be required to compute $\hat{\delta}_l$, and one inner product will be required to compute $\hat{W}_l^T A\hat{v}_n$. We will obtain $\hat{W}_{l-1}^T A\hat{v}_n$ without performing any inner products. To simplify the derivations, we will use the "monic" vectors v_n and w_n. All quantities involving the scaled vectors $\hat{v}_n$ and $\hat{w}_n$ can be obtained from the corresponding quantities involving v_n and w_n simply by scaling. Finally, we remark that, using a similar argument as in (5.1) below, one easily verifies that

$$W_l^T A v_n = V_l^T A^T w_n \quad \text{and} \quad W_{l-1}^T A v_n = V_{l-1}^T A^T w_n.$$

Therefore, the coefficients $\hat{\delta}_l^{-T}\hat{V}_l^T A^T \hat{w}_n$ and $\hat{\delta}_{l-1}^{-T}\hat{V}_{l-1}^T A^T \hat{w}_n$, which occur in the recursions for the left Lanczos vectors in (3.8) or (3.9), can be generated from $\hat{\delta}_l^{-1}\hat{W}_l^T A\hat{v}_n$ and $\hat{\delta}_{l-1}^{-1}\hat{W}_{l-1}^T A\hat{v}_n$, without computing any additional inner products.

First consider δ_l. Using (2.9) and the fact that polynomials in A commute, we deduce that

$$w_i^T v_j = w_1^T \Psi_i(A)\Psi_j(A)v_1 = w_1^T \Psi_j(A)\Psi_i(A)v_1 = w_j^T v_i. \tag{5.1}$$

This shows that the matrix δ_l is symmetric, and hence we only need to compute its upper triangle.

We will now show that once the diagonal and first superdiagonal of δ_l have been computed by inner products, the remaining upper triangle can be computed by recurrences. Let w_i and v_j be two vectors from the current block. Using (3.6) and the fact that the inner vectors from block l are orthogonal to the vectors from the previous block, we have

$$\begin{aligned} w_i^T v_j &= w_i^T (A v_{j-1} - \zeta_{j-1} v_{j-1} - \eta_{j-1} v_{j-2}) \\ &= (A^T w_i)^T v_{j-1} - \zeta_{j-1} w_i^T v_{j-1} - \eta_{j-1} w_i^T v_{j-2} \\ &= (w_{i+1} + \zeta_i w_i + \eta_i w_{i-1})^T v_{j-1} - \zeta_{j-1} w_i^T v_{j-1} - \eta_{j-1} w_i^T v_{j-2} \\ &= w_{i+1}^T v_{j-1} + \zeta_i w_i^T v_{j-1} + \eta_i w_{i-1}^T v_{j-1} - \zeta_{j-1} w_i^T v_{j-1} - \eta_{j-1} w_i^T v_{j-2}. \end{aligned}$$

Thus, $w_i^T v_j$ depends only on elements of δ_l from the previous two columns, and hence, with the exception of the diagonal and the first superdiagonal, can be computed without any additional inner products. Note that the recurrences and the orthogonality used in the above derivation are enforced numerically, and so computing $w_i^T v_j$ by the above recurrence should give the same results—up to roundoff—as computing the inner product directly.

We will now show how to compute $W_l^T A v_n$ with only one additional inner product, while $W_{l-1}^T A v_n$ can be obtained with no additional inner products. Consider $w_i^T A v_n$, for w_i a vector from either the current or the previous block. Then, we have

$$\begin{aligned} w_i^T A v_n &= (A^T w_i)^T v_n = (w_{i+1} + \zeta_i w_i + \eta_i w_{i-1})^T v_n \\ &= w_{i+1}^T v_n + \zeta_i w_i^T v_n + \eta_i w_{i-1}^T v_n. \end{aligned}$$

For $i < n_l - 1$, $W_{l-1}^T v_n = 0$, and hence $w_i^T A v_n = 0$. For $i = n_l - 1$, the above reduces to $w_{n_l-1}^T A v_n = w_{n_l}^T v_n$, which is computed as part of the first row of δ_l. For $n_l \le i < n_{l+1}$, all of the terms needed are available from δ_l. Finally, for the last vector in the current block, $i = n_{l+1} - 1$, we do not have $w_{n_{l+1}}^T v_n$, and hence have to compute it directly, thus requiring another inner product.

6. Numerical examples.

We have performed extensive numerical experiments with our implementation of the look-ahead Lanczos algorithm, both for eigenvalue problems and for the solution of linear systems. In this section, we present a few typical results of these experiments. Further numerical results are reported in [12] and [13].

Approximations to the eigenvalues of A can be obtained from the look-ahead Lanczos algorithm by computing some or all of the eigenvalues of the Lanczos matrix $\hat{H}^{(n)}$, the so-called *Ritz values*. In general, *spurious* approximate eigenvalues, caused by a loss of orthogonality among the Lanczos vectors, can occur among the Ritz values. This phenomenon is not due to the nonsymmetry of the matrix A; indeed, it also appears in the symmetric Lanczos process. We have used the heuristic due to Cullum and Willoughby [6] to identify and eliminate spurious Ritz values. Although this procedure was originally proposed for the scalar tridiagonal matrices generated by the standard Lanczos process, we also found it to work satisfactorily for the block tridiagonal matrices $\hat{H}^{(n)}$ produced by the look-ahead Lanczos algorithm. The eigenvalues of $\hat{H}^{(n)}$ were always computed using standard EISPACK routines.

For the solution of nonsingular linear systems

$$Ax = b, \tag{6.1}$$

we combine the look-ahead Lanczos algorithm with the QMR approach. More precisely, let $x_0 \in \mathbb{C}^N$ be any initial guess for (6.1) and choose the normalized starting residual vector

$$\hat{v}_1 := r_0/\rho_0, \qquad r_0 := b - Ax_0, \qquad \rho_0 := \|r_0\|,$$

as the first right Lanczos vector in Algorithm 3.1. The QMR method then generates approximate solutions to (6.1) defined by

$$x_n = x_0 + \hat{V}^{(n)} z_n, \qquad n = 1, 2, \ldots, \tag{6.2}$$

where z_n is the solution of the least-squares problem

$$\min_{z \in \mathbb{C}^n} \left\| \rho_0 e_1 - \hat{H}_e^{(n)} z \right\|, \qquad e_1 := [1 \quad 0 \quad \cdots \quad 0]^T \in \mathbb{R}^{(n+1)}. \tag{6.3}$$

We remark that, using (3.7), one easily verifies that the residual vector corresponding to the iterate (6.2) satisfies

$$r_n := b - Ax_n = \hat{V}^{(n+1)} \left(\rho_0 e_1 - \hat{H}_e^{(n)} z_n \right). \tag{6.4}$$

Thus the choice (6.3) of z_n just guarantees that the Euclidean norm of the coefficient vector in the representation (6.4) is minimal. For details and further properties of the QMR method, we refer to [13].

Example 6.1. This example is an eigenvalue problem, taken from [6]. Consider the differential operator

$$\begin{aligned} Lu := &-\frac{\partial}{\partial x}\left(e^{-xy}\frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(e^{xy}\frac{\partial u}{\partial y}\right) \\ &+ 20(x+y)\frac{\partial u}{\partial x} + 20\frac{\partial}{\partial x}\left((x+y)u\right) + \frac{1}{1+x+y}u \end{aligned} \tag{6.5}$$

on the unit square $(0,1) \times (0,1)$. We discretize (6.5) using centered differences on a 29×29 grid with mesh size $h = 1/30$. This leads to a nonsymmetric matrix of order $N = 900$. Unit vectors with random entries were used as starting vectors $\hat{v}_1$, $\hat{w}_1$ for Algorithm 3.1. The look-ahead Lanczos process was run for 320 steps, during which it built seven blocks of size 2. In Fig. 6.1, we plot the Ritz values (marked by "o") generated by the look-ahead Lanczos process after n = 40, 80, 160, 320 steps. We note that after 40 steps, the complex conjugate pair of Ritz values with maximal real part had converged to eigenvalues of A. After 80 steps, 12 Ritz values (all on the right edge of the spectrum) had converged, while after 160 steps the 30 Ritz values (24 on the right edge and 6 on the left edge of the spectrum) had converged to eigenvalues of A. In many cases, the standard and the look-ahead Lanczos procedures give similar results. In particular, for this example, the results obtained from both the standard and the look-ahead Lanczos algorithm match those reported in [6].

Example 6.2. This example is an eigenvalue problem, taken from [24], whose exact eigenvalues are known. Generally, problems of this type arise in modeling concentration waves in reaction and transport interaction of chemical solutions in a tubular reactor. The particular test problem used here corresponds to the so-called Brusselator wave model. This example was run for a matrix A of size $N = 100$. Again, unit vectors with random entries were used as starting vectors $\hat{v}_1$, $\hat{w}_1$. The look-ahead Lanczos algorithm needs $n = 112$ steps to obtain all the eigenvalues of A; it built two blocks of size 2. For this example, we have also run the standard Lanczos process without look-ahead, and computed the Ritz values after n = 100, 112, 120 steps. The denominators $\hat{w}_n^T\hat{w}_n$ were checked to exceed $\sqrt{\epsilon}$ in magnitude. In all three cases, some of the Ritz values obtained from the standard Lanczos process after deleting spurious eigenvalues do not correspond to any of the eigenvalues of A. In particular, the standard Lanczos process does not obtain the smallest eigenvalues of A even after 120 steps, and generates incorrect Ritz values, as shown in the plot. In Fig. 6.2, we plot the Ritz values generated by the look-ahead Lanczos process (marked by "o") and the Ritz values generated by the standard Lanczos process (marked by "+"), both after 120 steps.

Example 6.3. Here we consider a 6-cyclic matrix

$$A = \begin{bmatrix} I_1 & 0 & 0 & 0 & 0 & B_1 \\ B_2 & I_2 & 0 & 0 & 0 & 0 \\ 0 & B_3 & I_3 & 0 & 0 & 0 \\ 0 & 0 & B_4 & I_4 & 0 & 0 \\ 0 & 0 & 0 & B_5 & I_5 & 0 \\ 0 & 0 & 0 & 0 & B_6 & I_6 \end{bmatrix}, \tag{6.6}$$

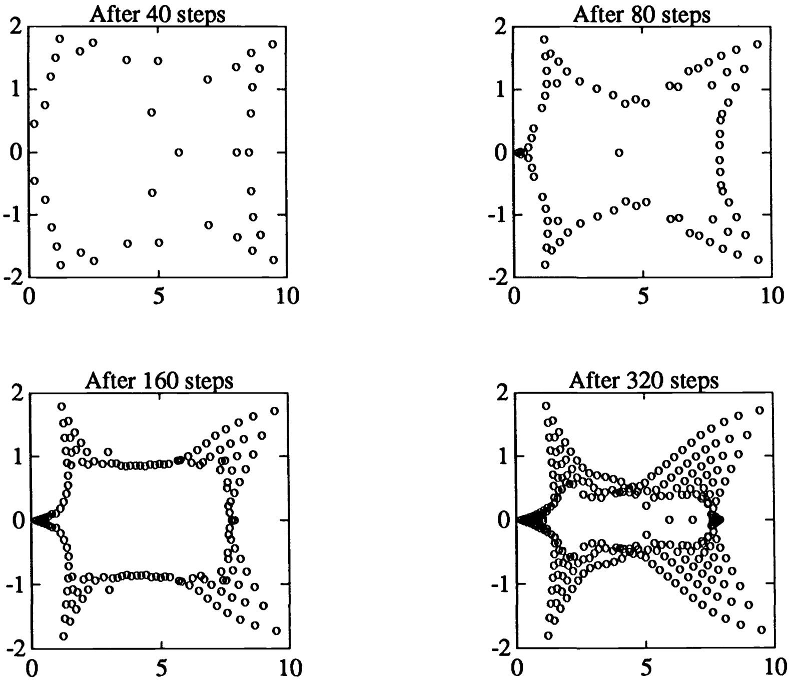


FIG. 6.1. *Ritz values for Example* 6.1, *obtained after* $n = 40, 80, 160, 320$ *steps of the look-ahead Lanczos algorithm.*

where the diagonal blocks I_1, I_2, I_3, I_4, I_5, and I_6 are identity matrices of size 827, 844, 827, 838, 831, and 838, respectively, so that A is a matrix of order $N = 5005$. This matrix arises in Markov chain modeling. For general p-cyclic matrices A of the form (6.6), Freund, Golub, and Hochbruck [10] have shown that work and storage of the look-ahead Lanczos process can be reduced to approximately $1/p$, as compared to arbitrary starting vectors, if v_1 and w_1 have only one nonzero block conforming to the block structure of A. Here, we have chosen

$$\hat{v}_1 = \begin{bmatrix} f_1 \\ 0 \end{bmatrix}, \quad \hat{w}_1 = \begin{bmatrix} g_1 \\ 0 \end{bmatrix},$$

where f_1, $g_1 \in \mathbb{R}^{827}$ have random entries. The look-ahead Lanczos algorithm generates blocks that alternately have sizes 1 and 5, starting with a block of size 1. In Fig. 6.3, we plot the Ritz values (marked by "o") generated by the look-ahead Lanczos process after $n = 40$, 80, 160, 320 steps. The standard Lanczos algorithm without look-ahead generates one Ritz value 1 in the first step, and then breaks down in the second step. Clearly, this example shows that the use of look-ahead is crucial if one wants to exploit the special structure of p-cyclic matrices.

Example 6.4. Here we solve a linear system (6.1) where A is the SHERMAN5 matrix taken from the Harwell–Boeing test collection of sparse matrices [8]. The matrix is of dimension $N = 3312$ and has 20793 nonzero elements. The right-hand side b in (6.1), as well as the first left Lanczos vector $\hat{w}_1$ were generated as different

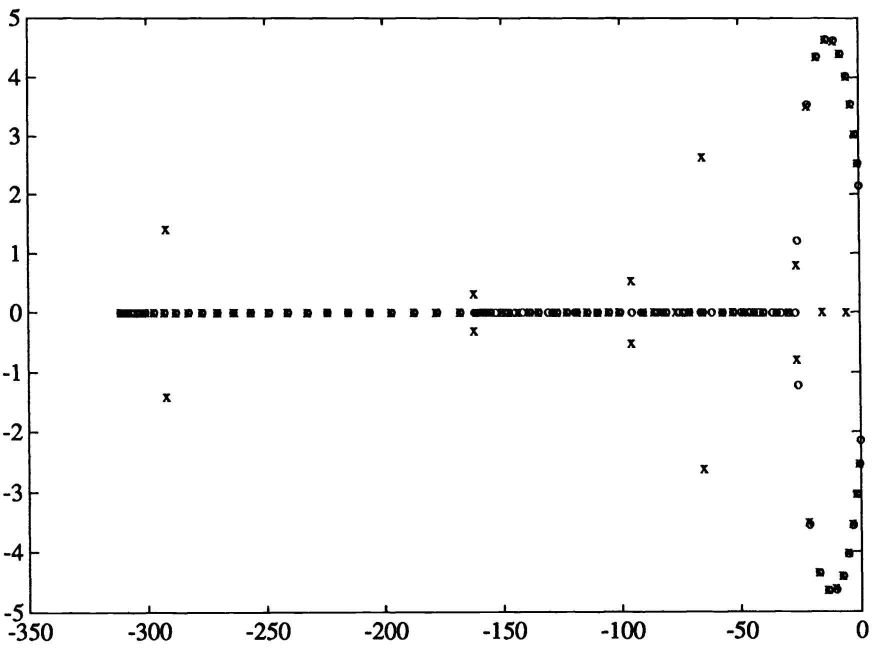


FIG. 6.2. *Ritz values (marked by "o," respectively "+") for Example* 6.2, *obtained from the look-ahead, respectively standard, Lanczos algorithm.*

unit vectors with random entries, and we set $x_0 = 0$ and $\hat{v}_1 = r_0 = b$. The QMR method takes $n = 1652$ steps to reduce the norm of the initial residual by a factor of 10^{-6}; see Fig. 6.4, where the residual norm $\|r_n\|$ is plotted versus n (solid line). The underlying look-ahead Lanczos algorithm built 34 blocks of size 2 and 7 blocks of size 3. We would like to stress that, for this example, look-ahead is crucial. Indeed, if look-ahead is turned off, then QMR based on the standard Lanczos algorithm does not converge. The corresponding stagnating residual norms (dashed line) are also depicted in Fig. 6.4.

Example 6.5. Here we consider the partial differential equation

$$Lu = f \quad \text{on} \quad (0,1) \times (0,1) \times (0,1), \tag{6.7}$$

where

$$\begin{aligned} Lu = &- \frac{\partial}{\partial x}\left(e^{xy}\frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(e^{xy}\frac{\partial u}{\partial y}\right) - \frac{\partial}{\partial z}\left(e^{xy}\frac{\partial u}{\partial z}\right) \\ &+ \beta(x+y+z)\frac{\partial u}{\partial x} + \left(\gamma + \frac{1}{1+x+y+z}\right)u, \end{aligned}$$

with Dirichlet boundary conditions $u = 0$. The right-hand side f is chosen such that

$$u = (1-x)(1-y)(1-z)\left(1-e^{-x}\right)\left(1-e^{-y}\right)\left(1-e^{-z}\right)$$

is the exact solution of (6.7). We set the parameters in (6.7) to $\beta = 30$ and $\gamma = -250$, and then we discretize (6.7) using centered differences on a uniform $15 \times 15 \times 15$

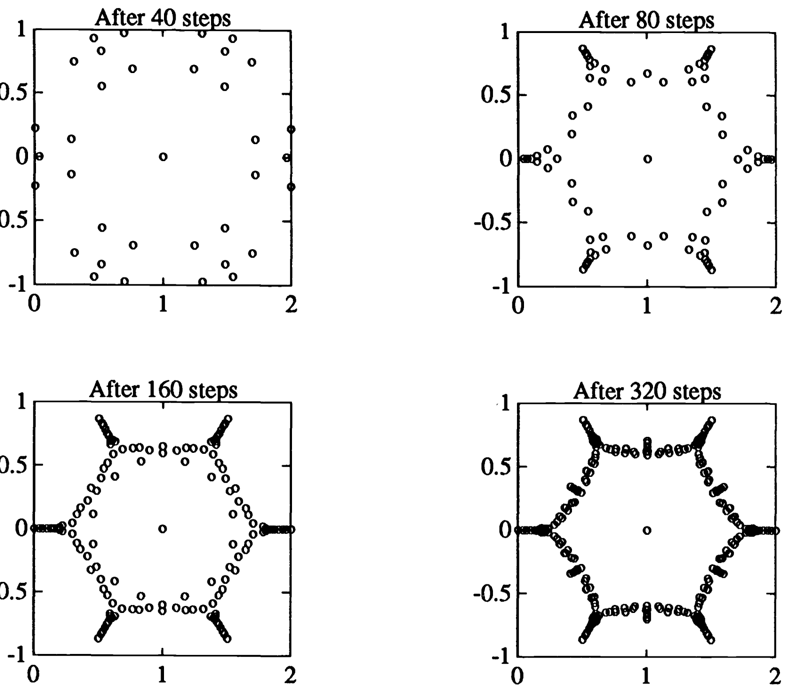


FIG. 6.3. *Ritz values for Example* 6.3, *obtained after* $n = 40, 80, 160, 320$ *steps of the look-ahead Lanczos algorithm.*

grid with mesh size $h = 1/16$. This leads to a linear system (6.1) with coefficient matrix A of order $N = 3375$ and 22275 nonzero elements. The QMR iteration was started with $x_0 = 0$. For the first pair of Lanczos vectors, we have chosen $\hat{w}_1 = \hat{v}_1 = b/\|b\|$ in Algorithm 3.1. The QMR approach takes $n = 149$ steps to reduce the norm of the initial residual by a factor of 10^{-6}; see Fig. 6.5, where the relative norm $\|r_n\| \,/\, \|r_0\|$ is plotted versus n (solid line). For this run, the underlying look-ahead Lanczos algorithm built three blocks of size 2. Next, we note that the matrix A is just the one used in Example 4.2. Recall that the look-ahead Lanczos algorithm based on the check (4.1) with tolerance tol$= \epsilon^{1/4} \approx$ 6.0E-08 encountered an artificial incurable breakdown. We also ran QMR based on this version of the look-ahead Lanczos algorithm, and the resulting convergence curve is shown as the dotted line in Fig. 6.5. Notice that, due to the artificial incurable breakdown, QMR does not converge in this case (cf. Fig. 4.2). Finally, we remark that QMR based on the standard Lanczos algorithm without look-ahead also converges for this example and gives a curve similar to the solid line in Fig. 6.5.

7. Conclusion.

We have proposed an implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. Our implementation can handle look-ahead steps of any length. Also, the proposed algorithm requires the same number of inner products as the standard Lanczos process without look-ahead. It was our intention to develop a robust algorithm which can be used in a black box.

FORTRAN 77 codes of our implementation of the look-ahead Lanczos algorithm

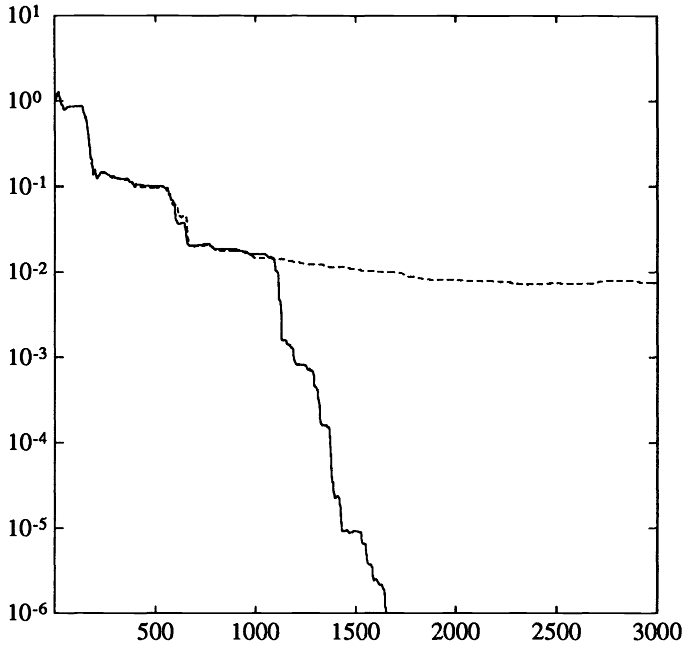


FIG. 6.4. *Residual norm* $\|r_n\|$ *plotted versus* n *for Example* 6.4.

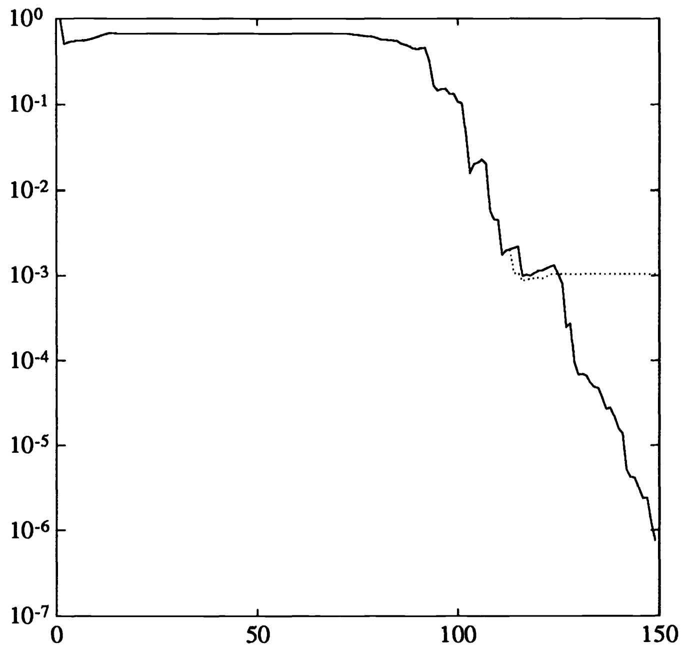


FIG. 6.5. *Relative residual norm* $\|r_n\|/\|r_0\|$ *plotted versus* n *for Example* 6.5.

and the QMR method are available electronically from the authors (na.freund@na-net.ornl.gov or na.nachtigal@na-net.ornl.gov).

Acknowledgments. Part of this work was done while the first and third authors were visiting the Interdisciplinary Project Center for Supercomputing at ETH Zürich, and we thank Martin Gutknecht for his warm hospitality. The authors are grateful to Marlis Hochbruck for her help with Example 6.3 and her careful reading of the manuscript.

REFERENCES

[1] D. L. Boley, S. Elhay, G. H. Golub, and M. H. Gutknecht, *Nonsymmetric Lanczos and finding orthogonal polynomials associated with indefinite weights*, Numer. Algorithms, 1 (1991), pp. 21–43.

[2] D. L. Boley and G. H. Golub, *The nonsymmetric Lanczos algorithm and controllability*, Systems Control Lett., 16 (1991), pp. 97–105.

[3] C. Brezinski, *Padé-Type Approximation and General Orthogonal Polynomials*, Birkhäuser, Basel, 1980.

[4] C. Brezinski, M. Redivo Zaglia, and H. Sadok, *A breakdown-free Lanczos type algorithm for solving linear systems*, Tech. Rep. ANO-239, Université des Sciences et Techniques de Lille Flandres-Artois, France, Jan. 1991.

[5] ———, *Avoiding breakdown and near-breakdown in Lanczos type algorithms*, Numer. Algorithms, 1 (1991), pp. 261–284.

[6] J. Cullum and R. A. Willoughby, *A practical procedure for computing eigenvalues of large sparse nonsymmetric matrices*, in Large Scale Eigenvalue Problems, J. Cullum and R. A. Willoughby, eds., North-Holland, Amsterdam, 1986, pp. 193–240.

[7] A. Draux, *Polynômes Orthogonaux Formels—Applications*, Lecture Notes in Mathematics 974, Springer-Verlag, Berlin, 1983.

[8] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[9] R. W. Freund, *Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 425–448.

[10] R. W. Freund, G. H. Golub, and M. Hochbruck, *Krylov subspace methods for non-Hermitian p-cyclic matrices*, Tech. Rep., RIACS, NASA Ames Research Center, Moffett Field, CA, in preparation.

[11] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal, *An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, Part* I, Tech. Rep. 90.45, RIACS, NASA Ames Research Center, Moffett Field, CA, November 1990.

[12] R. W. Freund and N. M. Nachtigal, *An implementation of the look-ahead Lanczos algorithm for non- Hermitian matrices, Part* II, Tech. Rep. 90.46, RIACS, NASA Ames Research Center, Moffett Field, CA, November 1990.

[13] ———, QMR: *A quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991) pp. 315–339.

[14] W. B. Gragg, *Matrix interpretations and applications of the continued fraction algorithm*, Rocky Mountain J. Math., 4 (1974), pp. 213–225.

[15] W. B. Gragg and A. Lindquist, *On the partial realization problem*, Linear Algebra Appl., 50 (1983), pp. 277–319.

[16] M. H. Gutknecht, *A completed theory of the unsymmetric Lanczos process and related algorithms, Part* I, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 594–639.

[17] ———, *A completed theory of the unsymmetric Lanczos process and related algorithms, Part* II, IPS Research Report No. 90–16, Zürich, Switzerland, September 1990.

[18] W. Joubert, *Lanczos methods for the solution of nonsymmetric systems of linear equations*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 926–943.

[19] S.-Y. Kung, *Multivariable and multidimensional systems: analysis and design*, Ph.D. thesis, Dept. of Electrical Engineering, Stanford University, Stanford, June 1977.

[20] 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.

[21] ———, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[22] T. A. MANTEUFFEL, *The Tchebychev iteration for nonsymmetric linear system*, Numer. Math. 28 (1977), pp. 307–327.

[23] B. N. PARLETT, *Reduction to tridiagonal form and minimal realizations*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 567–593.

[24] B. N. PARLETT AND Y. SAAD, *Complex shift and invert strategies for real matrices*, Linear Algebra Appl., 88/89 (1987), pp. 575–593.

[25] 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.

[26] E. L. STIEFEL, *Kernel polynomials in linear algebra and their numerical applications*, U.S. National Bureau of Standards, Applied Mathematics Series, 49 (1958), pp. 1–22.

[27] D. R. TAYLOR, *Analysis of the look ahead Lanczos algorithm*, Ph.D. thesis, Dept. of Mathematics, University of California, Berkeley, November 1982.

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

[29] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 159–184, January 1993

WAVELET-LIKE BASES FOR THE FAST SOLUTION OF SECOND-KIND INTEGRAL EQUATIONS*

B. ALPERT†, G. BEYLKIN‡, R. COIFMAN§, AND V. ROKHLIN¶

Abstract. A class of vector-space bases is introduced for the sparse representation of discretizations of integral operators. An operator with a smooth, nonoscillatory kernel possessing a finite number of singularities in each row or column is represented in these bases as a sparse matrix, to high precision. A method is presented that employs these bases for the numerical solution of second-kind integral equations in time bounded by $O(n \log^2 n)$, where n is the number of points in the discretization. Numerical results are given which demonstrate the effectiveness of the approach, and several generalizations and applications of the method are discussed.

Key words. wavelets, integral equations, sparse matrices, fast numerical algorithms

AMS(MOS) subject classifications. 42C15, 45L10, 65R10, 65R20

Introduction. Integral equations are a well-known mathematical tool for formulating physical problems. As a numerical tool they have several strengths (good conditioning, dimensionality reduction, and the ability to treat arbitrary regions), but have one overriding drawback: the high cost of working with the associated dense matrices. For a problem requiring an n-point discretization, the inverse of a dense $n \times n$ matrix must be applied to a vector. Even to apply the matrix itself to a vector requires order $O(n^2)$ operations; application of its inverse by a direct (noniterative) method requires order $O(n^3)$ operations. If an iterative method is employed, the number of iterations depends on the condition number of the problem and each iteration requires application of the $n \times n$ matrix. For large-scale problems, the resulting costs are often prohibitive.

In recent years a number of algorithms ([5], [10], [11], [14]) have been developed for the fast application of linear operators naturally expressible as dense matrices, the best known of which are the particle simulation algorithms developed by Greengard and Rokhlin [10]. These schemes combine low-order polynomial interpolation of the function, which defines the matrix elements with a divide-and-conquer strategy. They achieve (the equivalent of) order $O(n)$ application of a dense $n \times n$ matrix to a vector.

Over a somewhat longer period, mathematical bases have been constructed with certain *multiscale* properties. Families of functions $h_{a,b}$,

$$h_{a,b}(x) = |a|^{-1/2}\, h\left(\frac{x-b}{a}\right), \quad a, b \in \mathbf{R}, \quad a \neq 0,$$

derived from a single function h by dilation and translation, which form a basis for $L^2(\mathbf{R})$, are known as *wavelets* (Grossman and Morlet [12]). These families have

*Received by the editors December 20, 1990; accepted for publication (in revised form) October 7, 1991. This research was supported in part by Office of Naval Research grant N00014-86-0310, Defense Advanced Research Projects Agency grant DMS-9012751, IBM grant P00038437, and Department of Energy contract DE-AC03-76SF00098.

†Lawrence Berkeley Laboratory and Department of Mathematics, University of California, Berkeley, California 94720. Present address, National Institute of Standards and Technology, 325 Broadway, Boulder, Colorado 80303 (`alpert@bldr.nist.gov`).

‡Program in Applied Mathematics, University of Colorado in Boulder, Boulder, Colorado 80309-0526 (`beylkin@boulder.colorado.edu`).

§Department of Mathematics, Yale University, Box 2155 Yale Station, New Haven, Connecticut 06520 (`coifman@lom1.math.yale.edu`).

¶Departments of Mathematics and Computer Science, Yale University, Box 2158 Yale Station, New Haven, Connecticut 06520 (`rokhlin-vladimir@cs.yale.edu`).

been studied by many authors, resulting in constructions with a variety of properties. Meyer [13] constructed orthonormal wavelets for which $h \in C^{\infty}(\mathbf{R})$. Daubechies [7] constructed compactly supported wavelets with $h \in C^{k}(\mathbf{R})$ for arbitrary k, and [7] gives an overview and synthesis of the field.

A recent paper [6] establishes a connection between the fast numerical algorithms and the multiscale bases. It introduces the use of wavelets for the application of an integral operator to a function in $O(n \log n)$ operations, where n is the number of points in the discretization of the function. Alpert's thesis [4] gives an earlier report of the present work. Another paper [2] constructs a class of simple wavelet-like bases for $L^2[0,1]$ in which a variety of integral operators are sparse. In the present paper, rather than employ a wavelet basis for L^2, we construct a class of bases that transform the dense matrices resulting from the discretization of second-kind integral equations into sparse matrices. The $n \times n$ matrices resulting from an n-point discretization are transformed into matrices with order $O(n \log n)$ nonzero elements (to arbitrary finite precision). In these bases, the inverse matrices are also sparse, and are obtained in order $O(n \log^2 n)$ operations by a classical iterative method (due to Schulz [15]).

The method of this paper was developed with the aim of solving integral equations resulting from problems in potential theory, characterized by integral kernels that are smooth apart from diagonal singularities. In these problems, when high frequency modes have a significant presence in the given field (the right-hand side of the equation), a large number of points will be required in the discretization. The discretization must also be maintained for the integral operator, which dictates the need to solve a large-scale system of equations. If, instead, one has an integral operator with a globally smooth kernel, no large-scale system is required. In this case a direct method is entirely adequate, and preferable to the method given here.

In §1 we present the mathematical construction of the new bases. In §2 we briefly introduce Nyström's method for the solution of integral equations, and show how the wavelet-like bases result in sparse representation of integral operators and their inverses. We demonstrate that the Schulz method of matrix inversion is efficient in this context. In §3 we present the numerical algorithms for computation of the new bases, transformation of an integral operator into the bases, and computation of its inverse, and we analyze the time complexity of these algorithms. A variety of numerical examples are presented in §4 to demonstrate the effectiveness of the approach, and generalizations and applications are discussed in §5.

1. Wavelet-like bases.

1.1. Properties of the bases.

Given a set of n distinct points $S = \{x_1, x_2, \ldots, x_n\} \subset \mathbf{R}$ (the discretization) we construct an orthonormal basis for the n-dimensional space of functions defined on S. For simplicity, we assume that $n = k \cdot 2^l$, where k and l are positive integers, and that $x_1 < x_2 < \cdots < x_n$. The basis has two fundamental properties:

1. All but k basis vectors have k vanishing moments; and
2. The basis vectors are nonzero on different scales.

Figure 1 illustrates a matrix of basis vectors for $n = 128$ and $k = 4$. Each row represents one basis vector, with the dots depicting nonzero elements. The first k basis vectors are nonzero on $x_1, \ldots, x_{2k}$, the next k are nonzero on $x_{2k+1}, \ldots, x_{4k}$, and so forth. In all, one-half of the basis vectors are nonzero on $2k$ points from S, one-fourth are nonzero on $4k$ points, one-eighth are nonzero on $8k$ points, etc. Each of these $n/2 + n/4 + n/8 + \cdots + k = n - k$ basis vectors has k zero moments, i.e., if

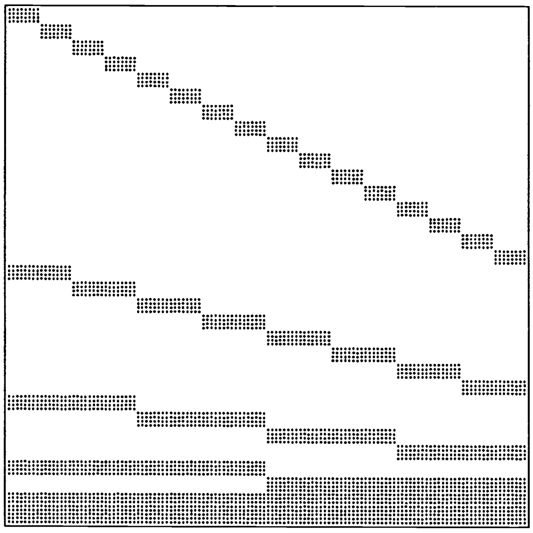


FIG. 1. *The matrix represents a wavelet-like basis for a discretization with* 128 *points, for* $k = 4$. *Each row denotes one basis vector, with the dots depicting nonzero elements. All but the final* k *rows have* k *vanishing moments.*

$b = \langle b_1, \ldots, b_n \rangle$ is one of these vectors, then

$$\sum_{i=1}^{n} b_i \cdot x_i{}^j = 0, \qquad j = 0, 1, \ldots, k-1.$$

The final k vectors result from orthogonalization of the moments $\langle x_1{}^j, x_2{}^j, \ldots, x_n{}^j \rangle$ for $j = 0, 1, \ldots, k-1$.

These properties of local support and vanishing moments lead to efficient representation of functions that are smooth except at a finite set of singularities. The projection of such a function on an element of this basis will be negligible unless the element is nonzero near one of the singularities. As a simple example, we consider the function $f(x) = \log(x)$ on the interval $[0, 1]$ with the uniform discretization $x_i = i/n$. A hand calculation shows that for any $c > 0$, f may be interpolated on the interval $[c, 2c]$ by a polynomial of degree 7 with error bounded by 4^{-9}, or roughly single precision accuracy. If we choose $k = 8$ in constructing the basis, f will be represented to this accuracy by the k basis vectors nonzero on $x_1, \ldots, x_{2k}$, the k basis vectors nonzero on $x_1, \ldots, x_{4k}$, and so forth, down to the k basis vectors nonzero on $x_1, \ldots, x_n$, in addition to the k orthogonalized moment vectors. The number of nonnegligible coefficients in the expansion of f in this basis grows logarithmically in n, the number of points of the discretization. Although this example is idealized, its behavior is representative of the general behavior of an analytic function near a singularity.

1.2. Construction of the bases. The conditions of "local" support and zero moments determine the basis vectors uniquely (up to sign) if we require somewhat more moments to vanish. Namely, out of the k vectors nonzero on $x_1, \ldots, x_{2k}$, we require that one have k vanishing moments, a second have $k+1$, a third have $k+2$, and so forth, and the kth have $2k-1$ vanishing moments. We place the same condition on the k basis vectors nonzero on $x_{2k+1}, \ldots, x_{4k}$, and so on, for each block of k basis vectors among the $n-k$ basis vectors with zero moments.

We construct the basis by construction of a finite sequence of bases (shown in Fig. 2), each obtained by a number of orthogonalizations. The first basis results from $n/(2k)$ Gram–Schmidt orthogonalizations of $2k$ vectors each. In particular, the vectors $\langle x_1{}^j, \ldots, x_{2k}{}^j \rangle$ for $j = 0, \ldots, 2k-1$ are orthogonalized, the vectors $\langle x_{2k+1}{}^j, \ldots, x_{4k}{}^j \rangle$ for $j = 0, \ldots, 2k-1$ are orthogonalized, and so forth, up to the vectors $\langle x_{n-2k+1}{}^j, \ldots, x_n{}^j \rangle$ for $j = 0, \ldots, 2k-1$, which are orthogonalized.

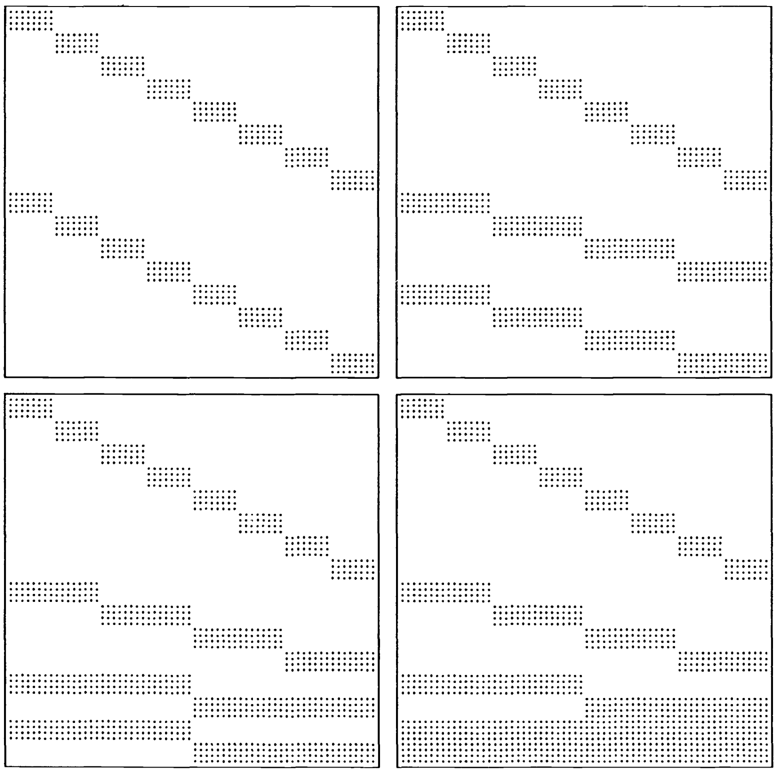


FIG. 2. *Each of the four matrices represents one basis, as in Fig. 1. The upper-left matrix is formed by orthogonalizing moment vectors on blocks of* $2k$ *points. The upper-right matrix is obtained from the upper-left matrix by premultiplying by an orthogonal matrix which is the identity on the upper half. Similarly, the lower matrices are obtained by further orthogonal transformations. The lower-right matrix represents the wavelet-like basis for* $n = 64, k = 4$.

Half of the n vectors of the first basis have at least k zero moments; in forming the second basis, these vectors are retained; the remaining $n/2$ basis vectors are transformed by an orthogonal transformation into basis vectors, each of which is nonzero on $4k$ of the points $x_1, \ldots, x_n$, and half of which have at least k vanishing moments. The orthogonal transformation results from $n/(4k)$ Gram–Schmidt orthogonalizations of $2k$ vectors each. Similarly, the third basis is obtained from the second basis by an orthogonal transformation that itself results from $n/(8k)$ Gram–Schmidt orthogonalizations of $2k$ vectors each. Before we can specify these orthogonalizations, we require some additional notation.

Suppose that V is a matrix whose columns $v_1, \ldots, v_{2k}$ are linearly independent. We define W=Orth(V) to be the matrix that results from the column-by-column Gram–Schmidt orthogonalization of V. Namely, denoting the columns of W by $w_1, \ldots, w_{2k}$, we have

$$\left.\begin{aligned} \text{linear span}\{w_1, \ldots, w_i\} &= \text{linear span}\{v_1, \ldots, v_i\}, \\ w_i^T w_j &= \delta_{ij}, \end{aligned}\right\} \quad i, j = 1, \ldots, 2k.$$

For a $2k \times 2k$ matrix V we let V^U and V^L denote two $k \times 2k$ matrices, V^U consisting of the upper k rows and V^L the lower k rows of V,

$$V = \begin{pmatrix} V^U \\ V^L \end{pmatrix}.$$

Now we proceed to the definition of the basis matrices. Given the set of points $S = \{x_1, \ldots, x_n\} \subset \mathbf{R}$ with $x_1 < \cdots < x_n$, where $n = k \cdot 2^l$, we define the $2k \times 2k$ moments matrices $M_{1,i}$ for $i = 1, \ldots, n/(2k)$ by the formula

$$M_{1,i} = \begin{pmatrix} 1 & x_{s_i+1} & \cdots & {x_{s_i+1}}^{2k-1} \\ 1 & x_{s_i+2} & \cdots & {x_{s_i+2}}^{2k-1} \\ \vdots & & & \vdots \\ 1 & x_{s_i+2k} & \cdots & {x_{s_i+2k}}^{2k-1} \end{pmatrix}, \tag{1}$$

where $s_i = (i-1)2k$. The first basis matrix U_1 is the $n \times n$ matrix given by the formula

$$U_1 = \begin{pmatrix} {U_{1,1}}^L & & & \\ & {U_{1,2}}^L & & \\ & & \ddots & \\ & & & {U_{1,n_1}}^L \\ {U_{1,1}}^U & & & \\ & {U_{1,2}}^U & & \\ & & \ddots & \\ & & & {U_{1,n_1}}^U \end{pmatrix},$$

where ${U_{1,i}}^T = \text{Orth}(M_{1,i})$ and $n_1 = n/(2k)$. The second basis matrix is U_2U_1, with U_2 defined by the formula

$$U_2 = \begin{pmatrix} I_{n/2} & \\ & U_2' \end{pmatrix},$$

where I_m is the $m \times m$ identity matrix and the $n/2 \times n/2$ matrix U_2' is given by the formula

$$U_2' = \begin{pmatrix} {U_{2,1}}^L & & & \\ & {U_{2,2}}^L & & \\ & & \ddots & \\ & & & {U_{2,n_2}}^L \\ {U_{2,1}}^U & & & \\ & {U_{2,2}}^U & & \\ & & \ddots & \\ & & & {U_{2,n_2}}^U \end{pmatrix},$$

where $n_2 = n/(4k)$, ${U_{2,i}}^T = \text{Orth}\,(M_{2,i})$, and the $2k \times 2k$ matrix $M_{2,i}$ is given by

$$M_{2,i} = \begin{pmatrix} {U_{1,2i-1}}^U M_{1,2i-1} \\ {U_{1,2i}}^U M_{1,2i} \end{pmatrix}$$

for $i = 1, \ldots, n/(4k)$. In general, the jth basis matrix, for $j = 2, \ldots, \log_2(n/k)$, is $U_j \cdots U_1$, with U_j defined by the formula

$$U_j = \begin{pmatrix} I_{n-n/2^{j-1}} & \\ & U_j' \end{pmatrix},$$

where U_j' is given by the formula

$$U_j' = \begin{pmatrix} {U_{j,1}}^L & & & \\ & {U_{j,2}}^L & & \\ & & \ddots & \\ & & & {U_{j,n_j}}^L \\ {U_{j,1}}^U & & & \\ & {U_{j,2}}^U & & \\ & & \ddots & \\ & & & {U_{j,n_j}}^U \end{pmatrix},$$

where $n_j = n/(2^j k)$; $U_{j,i}$ is given by

(2) $$ {U_{j,i}}^T = \text{Orth}(M_{j,i});$$

and $M_{j,i}$ is given by

(3) $$M_{j,i} = \begin{pmatrix} {U_{j-1,2i-1}}^U M_{j-1,2i-1} \\ {U_{j-1,2i}}^U M_{j-1,2i} \end{pmatrix}$$

for $i = 1, \ldots, n/(2^j k)$. The final basis matrix $U = U_l \cdots U_1$, where $l = \log_2(n/k)$, represents the wavelet-like basis of parameter k on $x_1, \ldots, x_n$. Note that the matrices

U and U_j are of dimension $n \times n$, U'_j is $n/2^{j-1} \times n/2^{j-1}$, $U_{j,i}$ and $M_{j,i}$ are $2k \times 2k$, and $U^L_{j,i}$ and $U^U_{j,i}$ are $k \times 2k$.

Remark 1.1. The definitions given for the basis matrices are mathematical definitions only; in a numerical procedure, considerable roundoff error would be introduced by the orthogonalizations defined above. In the actual implementation, the matrices $M_{j,i}$ are shifted and scaled, resulting in a numerically stable procedure that is equivalent to the above definitions (in exact arithmetic). Details of this procedure are provided in §3.

It is apparent that the application of the matrix U to an arbitrary vector of length n may be accomplished in order $O(n)$ operations by the application of $U_1, \ldots, U_l$ in turn. Similarly, $U^{-1} = U^T$ may be applied to a vector in order $O(n)$ operations. Certain dense matrices, in particular those arising from integral operators, are sparse in the basis of U and their similarity transformations can be computed in $O(n \log n)$ operations. These techniques are developed in the following sections.

Figure 3 illustrates the vectors of one basis from this class.

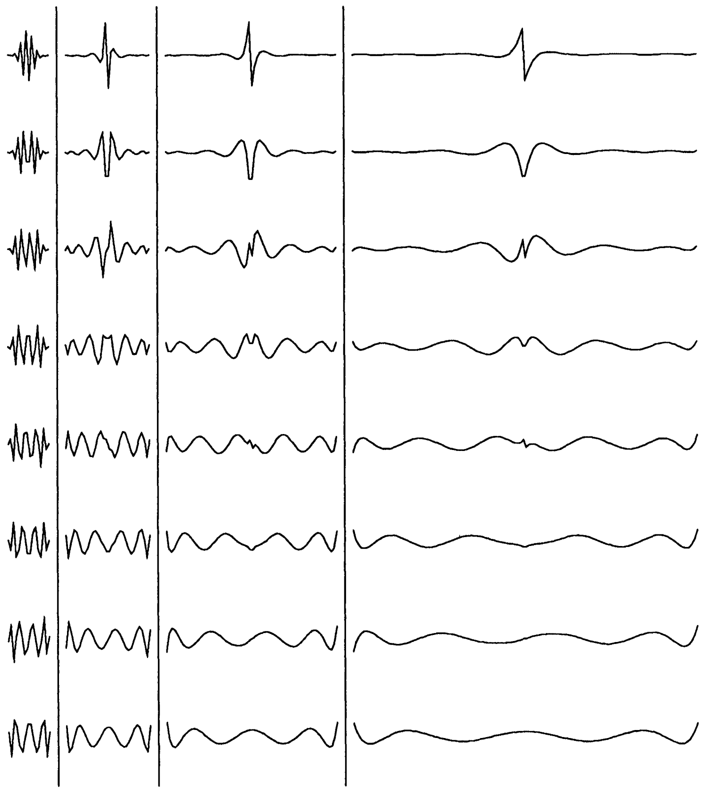


FIG. 3. *Basis vectors on four scales are shown for the basis where $n = 128$, points $x_1, \ldots, x_n$ are equispaced, and $k = 8$. The first column of vectors consists of rows 1–8 of U, the second column consists of rows 65–72, etc. Note that half of the vectors are odd and half are even functions, and that the odd ones are generally discontinuous at their center.*

2. Second-kind integral equations.

2.1. Nyström method. A linear Fredholm integral equation of the second kind is an expression of the form

$$f(x) - \int_a^b K(x,t)\, f(t)\, dt = g(x), \tag{4}$$

where the kernel K is in $L^2[a,b]^2$ and the unknown f and right-hand side g are in $L^2[a,b]$. We use the symbol $\mathcal{K}$ to denote the integral operator of (4), which is given by the formula

$$(\mathcal{K}f)(x) = \int_a^b K(x,t)\, f(t)\, dt,$$

for all $f \in L^2[a,b]$ and $x \in [a,b]$. Then (4), written in operator form, is

$$(I - \mathcal{K})f = g. \tag{5}$$

The Nyström, or *quadrature*, method for the numerical solution of integral equations approximates the integral operator $\mathcal{K}$ by the finite-dimensional operator R, characterized by points $x_1, x_2, \ldots, x_n \in [a,b]$ and weights $w_1, w_2, \ldots, w_n \in \mathbf{R}$, and given by the formula

$$(Rf)(x) = \sum_{j=1}^{n} w_j\, K(x, x_j)\, f(x_j), \tag{6}$$

for all $f \in L^2[a,b]$ and $x \in [a,b]$. Substitution of R for $\mathcal{K}$ in (5), combined with the requirement that the resulting equation holds for $x = x_1, x_2, \ldots, x_n$, yields the following system of n equations in the n unknowns $f_1, f_2, \ldots, f_n$:

$$f_i - \sum_{j=1}^{n} w_j\, K(x_i, x_j)\, f_j = g(x_i), \qquad i = 1, \ldots, n. \tag{7}$$

The approximation $\langle f_1, \ldots, f_n \rangle$ to the solution f of (4) may be extended to all $x \in [a,b]$ by the natural formula

$$f_R(x) = g(x) + \sum_{i=1}^{n} w_j\, K(x, x_j)\, f_j, \tag{8}$$

which satisfies $f_R(x_i) = f_i$ for $i = 1, \ldots, n$. How large is the error $e_R = f - f_R$ of the approximate solution? We follow the derivation by Delves and Mohamed in [8]. Rewriting (7) in operator form, we have

$$(I - R)f_R = g, \tag{9}$$

and combining (5) and (9) yields $(I - \mathcal{K})e_R = (\mathcal{K} - R)f_R$. Provided that $(I - \mathcal{K})^{-1}$ exists, we obtain the error bound

$$\|e_R\| \leq \|(I - \mathcal{K})^{-1}\| \cdot \|(\mathcal{K} - R)f_R\|. \tag{10}$$

The error depends, therefore, on the conditioning of the original integral equation, as is apparent from the term $\|(I - \mathcal{K})^{-1}\|$, and on the fidelity of the quadrature R to the integral operator $\mathcal{K}$. It is not necessary that $\|\mathcal{K} - R\|$ be small, rather merely that R approximate $\mathcal{K}$ well near the solution f. Quadrature rules that have this property, but that are defined only on the points $x_1, \ldots, x_n$, are developed in [3]. In these rules the quadrature weights w_j of (6) become w_{ij}, which depend on the point of definition x_i, for $i = 1, \ldots, n$. The quadrature rules converge rapidly for kernels with singularities of known location and type. These rules are used below in the numerical examples of §4.

2.2. Sparse representation of integral operators. We concern ourselves here with kernels $K = K(x,t)$, which are analytic except at $x = t$, where they possess an integrable singularity. We initially discretize the integral operator $\mathcal{K}$ using a simple equispaced quadrature. Given $n \geq 2$, we define points $x_1, \ldots, x_n$ to be equispaced on the interval $[a, b]$,

$$x_i = a + (i-1)(b-a)/(n-1), \tag{11}$$

and define the elements T_{ij} of the $n \times n$ matrix T by the formula

$$T_{ij} = \begin{cases} \frac{1}{n-1} K(x_i, x_j), & i \neq j, \\ 0, & i = j. \end{cases} \tag{12}$$

Note that the matrix $T = T(n)$ corresponds to a primitive, trapezoid-like quadrature discretization of the integral operator $\mathcal{K}$. The matrix T possesses the same smoothness properties as the kernel $K(x,t)$. Transformation of T by the bases developed in §1 produces a matrix that is sparse, to high precision. The number of elements is effectively bounded by order $O(n \log n)$.

When the matrix representing the quadrature corrections developed in [3] is added to T, producing high-order convergence to the integral operator, this complexity bound remains valid.

The matrix T, transformed by the orthogonal $n \times n$ matrix U, can be decomposed into the sum of a sparse matrix and a matrix with small norm. Given $\epsilon > 0$, there exists $c_\epsilon > 0$, independent of n, such that the transformed matrix can be written in the form

$$UTU^T = V + E,$$

where the number of elements in $V = V(n)$ is bounded by $c_\epsilon n \log n$ and $E = E(n)$ is small: $\|E\| < \epsilon \|T\|$. We do not prove this assertion here; the proof parallels the proofs of similar statements in [2], but is somewhat more tedious.

2.3. Solution via Schulz method. The sparse matrix representing the integral operator also has a sparse inverse, which can be computed rapidly.

Schulz's method [15] is an iterative, quadratically convergent algorithm for computing the inverse of a matrix. Its performance is characterized in the following lemma.

LEMMA 2.1. *Suppose that A is an invertible matrix, X_0 is the matrix given by $X_0 = A^H/\|A^H A\|$, and for $m = 0, 1, 2, \ldots$ the matrix X_{m+1} is defined by the recursion*

$$X_{m+1} = 2X_m - X_m A X_m.$$

Then X_{m+1} satisfies the formula

$$I - X_{m+1}A = (I - X_m A)^2. \tag{13}$$

Furthermore, $X_m \to A^{-1}$ *as* $m \to \infty$ *and for any* $\epsilon > 0$ *we have*

$$\|I - X_m A\| < \epsilon \quad \textit{provided} \quad m \geq 2\log_2 \kappa(A) + \log_2 \log(1/\epsilon), \tag{14}$$

where $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ *is the condition number of* A *and the norm is given by* $\|A\| = (\text{largest eigenvalue of } A^H A)^{1/2}$.

Proof. Equation (13) is obtained directly from the definition of X_{m+1}. Bound (14) is equally straightforward. Noting that $A^H A$ is symmetric positive-definite and letting λ_0 denote the smallest, and λ_1 the largest, eigenvalue of $A^H A$ we have

$$\begin{aligned} \|I - X_0 A\| &= \left\| I - \frac{A^H A}{\|A^H A\|} \right\| \\ &= 1 - \lambda_0/\lambda_1 \\ &= 1 - \kappa(A)^{-2}. \end{aligned} \tag{15}$$

From (13) we obtain $I - X_m A = (I - X_0 A)^{2^m}$, which in combination with (15) and simple manipulation yields bound (14). □

The Schulz method is a notably simple scheme for matrix inversion and its convergence is extremely rapid. It is rarely used, however, because it involves matrix–matrix multiplications on each iteration; for most problem formulations, this process requires order $O(n^3)$ operations for an $n \times n$ matrix. We observe, however, that a sparse matrix, possessing a sparse inverse, whose iterates X_n are also sparse, may be rapidly inverted using the Schulz method. We have seen above that a discretized integral operator $I - T$, similarity-transformed to the representation $A = I - UTU^T$, has only order $O(n \log n)$ elements (to finite precision). In addition, $A^T A$ and $(A^T A)^m$ are similarly sparse. This property enables us to employ the Schulz algorithm to compute A^{-1} in order $O(n \log^2 n)$ operations.

2.4. Oscillatory coefficients. We now consider a somewhat more general class of integral equations, in which the integral operator is given by the formula

$$(D\mathcal{K}f)(x) = p(x) \int_a^b K(x,t)\, f(t)\, dt,$$

where the kernel K is assumed to be smooth, but the coefficient function p can be oscillatory. In particular, we only restrict p to be positive. In terms of generality, these problems lie between the problems with smooth kernels (and constant coefficient) and those with arbitrary oscillatory kernels.

Writing the corresponding integral equation in operator form, we obtain the equation

$$(I - D\mathcal{K})f = g. \tag{16}$$

Although D is a diagonal operator, and $\mathcal{K}$ is smooth, it is clear that the discretization of the operator $D\mathcal{K}$ will not be a sparse matrix in wavelet coordinates. In this framework, it would appear that the construction of this paper is inapplicable. If we instead consider the operator $D^{1/2}\mathcal{K}D^{1/2}$, in which oscillations in the rows match those in the columns, it becomes clear that the construction of §1 can be revised. Rather than constructing basis functions orthogonal to low-order polynomials x^j, we can construct them to be orthogonal to $p(x)^{1/2}\, x^j$. The sole revision in our definition of basis matrices $U_1, \ldots, U_l$ is to replace the definition (1) of the moments matrices $M_{1,i}$ for $i = 1, \ldots, n/(2k)$, by the new definition

$$M_{1,i} = \begin{pmatrix} p_{s_i+1} & 0 & \cdots & 0 \\ 0 & p_{s_i+2} & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & 0 & p_{s_i+2k} \end{pmatrix} \begin{pmatrix} 1 & x_{s_i+1} & \cdots & {x_{s_i+1}}^{2k-1} \\ 1 & x_{s_i+2} & \cdots & {x_{s_i+2}}^{2k-1} \\ \vdots & & & \vdots \\ 1 & x_{s_i+2k} & \cdots & {x_{s_i+2k}}^{2k-1} \end{pmatrix},$$

where $s_i = (i-1)2k$ and $p_j = p(x_j)^{1/2}$.

Now the integral equation (16) can be transformed to the equation

$$(I - D^{1/2}\mathcal{K}D^{1/2})(D^{-1/2}f) = (D^{-1/2}g),$$

which is discretized to a system that is sparse in the revised wavelet-like coordinates. The inverse matrix is also sparse.

3. Numerical algorithms.

In §1 we defined a class of bases for functions defined on $\{x_1, \ldots, x_n\}$, and in §2 we showed that, to finite precision, second-kind integral operators and their inverses are asymptotically sparse in these bases. In this section we present procedures for computation of the bases, discretized integral operators in these bases, and the inverses of these operators. In §4 we give some numerical examples based on our implementations of these procedures.

The computation of the new bases is discussed next, followed by a discussion of the transformation of the integral operators to the new bases. We defer discussion of the computation of the inverses, sketched above, to §3.3, which contains detailed descriptions of all of the algorithms. Finally, §3.4 gives the complexity analysis for the algorithms.

3.1. Computation of wavelet-like bases.

It was mentioned in §1 that the mathematical definition of $U_1, \ldots, U_l$, if used directly, would result in a numerical procedure that would create large roundoff errors. A correct procedure is obtained by shifting and scaling the matrices $M_{j,i}$ defined there.

For a pair of numbers $(\mu, \sigma) \in \mathbf{R} \times (\mathbf{R}\backslash\{0\})$ we define a $2k \times 2k$ matrix $S(\mu, \sigma)$ whose (i,j)th element is the binomial term

$$S(\mu,\sigma)_{i,j} = \binom{j-1}{i-1} \frac{(-\mu)^{j-i}}{\sigma^{j-1}} \tag{17}$$

for $i \le j$, and $S(\mu,\sigma)_{i,j} = 0$ otherwise. The matrix $S(\mu,\sigma)$ is upper-triangular and nonsingular, and its inverse is given by the formula

$$S(\mu,\sigma)^{-1} = S(-\mu/\sigma, 1/\sigma). \tag{18}$$

Furthermore, the product formula

$$S(\mu_1,\sigma_1)S(\mu_2,\sigma_2) = S(\mu_1 + \mu_2\sigma_1, \sigma_1\sigma_2) \tag{19}$$

is easily verified.

We define $M'_{j,i}$ for $j = 1, \ldots, l$ and $i = 1, \ldots, n/(2^j k)$ by the formula

$$M'_{j,i} = M_{j,i}S(\mu_{j,i}, \sigma_{j,i}), \tag{20}$$

where $\mu_{j,i} = (x_{1+(i-1)k2^j} + x_{ik2^j})/2, \sigma_{j,i} = (x_{ik2^j} - x_{1+(i-1)k2^j})/2$, and the matrix $M_{j,i}$ is defined by (1) and (3) in §1. The matrix $U_{j,i}$ is given by the formula

$$U_{j,i}{}^T = \mathrm{Orth}(M'_{j,i}), \tag{21}$$

which is equivalent to the definition given by (2). This equivalence immediately follows from the fact that $S(\mu, \sigma)$ is upper-triangular and nonsingular.

The matrices $M'_{1,i}$ for $i = 1, \ldots, n/(2k)$ are actually computed by the formula

$$M'_{1,i} = \begin{pmatrix} 1 & \frac{x_{s_i+1}-\mu_{1,i}}{\sigma_{1,i}} & \cdots & \left(\frac{x_{s_i+1}-\mu_{1,i}}{\sigma_{1,i}}\right)^{2k-1} \\ 1 & \frac{x_{s_i+2}-\mu_{1,i}}{\sigma_{1,i}} & \cdots & \left(\frac{x_{s_i+2}-\mu_{1,i}}{\sigma_{1,i}}\right)^{2k-1} \\ \vdots & & & \vdots \\ 1 & \frac{x_{s_i+2k}-\mu_{1,i}}{\sigma_{1,i}} & \cdots & \left(\frac{x_{s_i+2k}-\mu_{1,i}}{\sigma_{1,i}}\right)^{2k-1} \end{pmatrix}, \tag{22}$$

where $s_i = (i-1)2k$. Likewise, the matrices $M'_{j,i}$ for $j = 2, \ldots, l$ and $i = 1, \ldots, n/(2^j k)$ are computed by the formula

$$M'_{j,i} = \begin{pmatrix} U_{j-1,2i-1}{}^U M'_{j-1,2i-1} S^1_{j,i} \\ U_{j-1,2i}{}^U M'_{j-1,2i} S^2_{j,i} \end{pmatrix}, \tag{23}$$

where $S^1_{j,i}$ and $S^2_{j,i}$ are defined by the formulae

$$S^1_{j,i} = S(\mu_{j-1,2i-1}, \sigma_{j-1,2i-1})^{-1} S(\mu_{j,i}, \sigma_{j,i}), \tag{24}$$

$$S^2_{j,i} = S(\mu_{j-1,2i}, \sigma_{j-1,2i})^{-1} S(\mu_{j,i}, \sigma_{j,i}). \tag{25}$$

Application of the inverse and product rules given in (18) and (19) to (24) and (25) yields formulae by which $S^1_{j,i}$ and $S^2_{j,i}$ can be computed:

$$S^1_{j,i} = S((\mu_{j,i} - \mu_{j-1,2i-1})/\sigma_{j-1,2i-1},\ \sigma_{j,i}/\sigma_{j-1,2i-1}), \tag{26}$$

$$S^2_{j,i} = S((\mu_{j,i} - \mu_{j-1,2i})/\sigma_{j-1,2i},\ \sigma_{j,i}/\sigma_{j-1,2i}). \tag{27}$$

The matrices $M'_{j,i}$ given by (22) and (23) are easily seen to be mathematically equivalent to those defined by (20); nonetheless, computation of $M'_{j,i}$ using (22) and (23) avoids the large roundoff errors that would otherwise result.

3.2. Transformation to wavelet-like bases. We assume that for equispaced points $x_1, \ldots, x_n$ (defined in (11)) and some k, the orthogonal matrices $U_1, \ldots, U_l$ defined in §1 have been computed ($l = \log_2(n/k)$). We now present a procedure for computation of UTU^T, where $U = U_l \cdots U_1$ and T is the discretized integral operator defined in (12).

3.2.1. Simple example. We begin with a simplified example in which T is replaced by an $n \times n$ matrix V of rank k whose elements V_{ij} are defined by the equation

$$V_{ij} = \sum_{r=1}^{k} \sum_{s=1}^{k} \Lambda_{rs} x_i{}^{r-1} x_j{}^{s-1}, \qquad i, j = 1, \ldots, n.$$

Each row and each column of V contains elements that are the values of a polynomial of degree $k-1$. The matrix V can be written as $V = P^T \Lambda P$, where the elements of the $k \times n$ matrix P are defined by $P_{ij} = {x_j}^{i-1}$ and Λ is the $k \times k$ matrix with elements Λ_{ij}. Recalling that the last k rows of the basis matrix U consist of an orthogonalization of the moment vectors $\langle {x_1}^j, \ldots, {x_n}^j \rangle$ for $j = 0, \ldots, k-1$, we can rewrite V as $V = (P')^T \Lambda' P'$. Here the $k \times n$ matrix P' consists of the last k rows of U and Λ' is a new $k \times k$ matrix with elements Λ'_{ij}.

By the orthogonality of U, it is clear that the $n \times n$ matrix $UVU^T = U(P')^T \Lambda' P' U^T$ consists entirely of zero elements except the $k \times k$ submatrix in the lower-right corner, which is the matrix Λ'. Given a function to compute elements of the $n \times n$ matrix V, the matrix Λ' can be computed in time independent of n by using a $k \times k$ extract of values from V. We form the $k \times k$ matrix V' with elements V'_{ij} defined by the formula

$$V'_{ij} = V_{in/k,jn/k}, \qquad i,j = 1, \ldots, k. \tag{28}$$

Then $V' = (P'')^T \Lambda' P''$, where P'' is the $k \times k$ extract of P' with elements given by $P''_{ij} = P'_{i,jn/k}$. Thus we obtain

$$\Lambda' = ((P'')^T)^{-1} V' (P'')^{-1} \tag{29}$$

from P'' and V' readily in $O(k^3)$ operations, and we have obtained UVU^T.

3.2.2. General case. The integral operator matrix T is, of course, not of low rank, but it can be divided into submatrices, each approximately of rank k (see Fig. 4). The submatrices near the main diagonal are of size $k \times k$, those next removed are $2k \times 2k$, and so forth up to the largest submatrices, of size $n/4 \times n/4$. The total number of submatrices is proportional to n/k. Given an error tolerance $\epsilon > 0$, k may be chosen (independently of n) so that each submatrix of T, say T^i, may be written as a sum, $T^i = V^i + E^i$, where the elements of V^i are given by a polynomial of degree $k-1$ and $\|E^i\| < \epsilon \|T^i\|$.

The simplified example, in which the matrix to be transformed is of rank k, is now applicable. Each submatrix of T is treated as a matrix of rank k and is transformed to the new coordinates (for its own scale) in order $O(k^3)$ operations. To make this precise, we write $T = T_0 + \cdots + T_{l-2}$ where T_i consists of the submatrices of size $2^i k \times 2^i k$. For each i, the submatrices of T_i may be interpolated by rank k submatrices, as indicated by the extract of (28), to obtain matrices V_i. Thus $T_i = V_i + E_i$, where $\|E_i\|$ is small. In the simplified example above, we have shown that the transformed matrices

$$\begin{aligned} W_0 &= V_0, \\ W_1 &= U_1 V_1 {U_1}^T, \\ W_2 &= U_2 U_1 V_2 {U_1}^T {U_2}^T, \\ &\vdots \\ W_{l-2} &= U_{l-2} \cdots U_1 V_{l-2} {U_1}^T \cdots {U_{l-2}}^T \end{aligned} \tag{30}$$

can be computed by many applications of (29), all in order $O(nk^2)$ operations. This estimate follows from the fact that there are $O(n/k)$ submatrices, each of which is transformed in $O(k^3)$ operations. Now we define $n \times n$ matrices $R_0, \ldots, R_l$ recursively:

$$R_i = \begin{cases} W_0, & i = 0, \\ U_i R_{i-1} {U_i}^T + W_i, & i \geq 1 \end{cases} \tag{31}$$

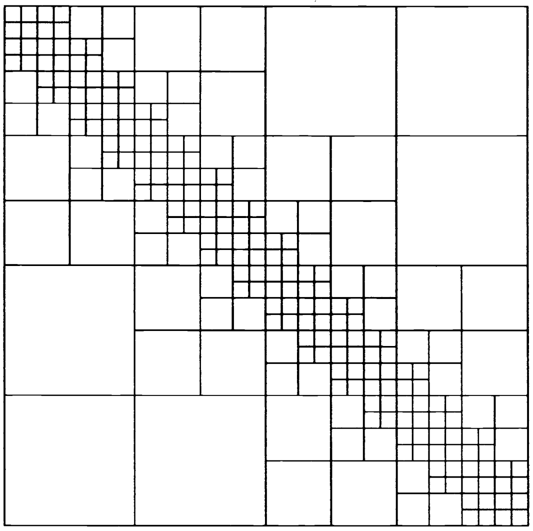


FIG. 4. *The matrix represents a discretized integral operator with a kernel that is singular along the diagonal. The matrix is divided into submatrices of rank k (to high precision) and transformed to a sparse matrix with order $O(n \log n)$ elements. Here $n/k = 32$.*

(here $W_{l-1} = W_l = 0$). Then R_l contains the final result, $R_l = U(T - E)U^T$, where $E = E_0 + \cdots + E_{l-2}$.

The matrix–matrix products in the definition of $R_0, \ldots, R_l$ can be computed directly, since the factors and the products contain no more than $O(n \log n)$ elements. A simple implementation with standard sparse matrix structures results in a total operation count of order $O(n \log^2 n)$, but an implementation using somewhat more elaborate data structures, in which repetitive handling of data is avoided, requires only order $O(n \log n)$ operations.

Computation using the result R_l is made more efficient by removing the elements of R_l which can be neglected, within the precision with which R_l approximates UTU^T. For a given precision ϵ, we discard a matrix E' by eliminating elements from R_l below a threshold τ. The threshold depends on the choice of norm; in our implementation, we use the row-sum norm

$$\|A\| = \max_i \sum_{j=1}^{n} |A_{ij}|,$$

for an $n \times n$ matrix A. The element threshold

$$\tau = \frac{\epsilon}{n}\|T\| \tag{32}$$

clearly results in a discarded matrix E' with $\|E'\| < \epsilon\|T\|$.

3.3. Detailed descriptions of algorithms.

PROCEDURE TO COMPUTE $U_1, \ldots, U_l$

Comment [Input to this procedure consists of the number of points n, the number of zero moments k, and the points $x_1, \ldots, x_n$. Output is the matrices $U_{j,i}$ for $j = 1, \ldots, l$ and $i = 1, \ldots, n/(2^j k)$, which make up the matrices $U_1, \ldots, U_l$ (note $l = \log_2(n/k)$).]

Step 1

Compute the shifted and scaled moments matrices $M'_{1,i}$ for $i = 1, \ldots, n/(2k)$ according to (22).

Step 2

Compute $U_{1,i}$ from $M'_{1,i}$ by (21) using Gram–Schmidt orthogonalization for $i = 1, \ldots, n/(2k)$.

Step 3

Comment [Compute $M'_{j,i}$ and $U_{j,i}$ for $j = 2, \ldots, l$ and $i = 1, \ldots, n/(2^j k)$.]

```
do $j = 2, \ldots, l$
   do $i = 1, \ldots, n/(2^j k)$
      Compute $U_{j-1,2i-1}{}^U M'_{j-1,2i-1}$ and $U_{j-1,2i}{}^U M'_{j-1,2i}$.
     Compute $S^1_{j,i}$ by (26) and $S^2_{j,i}$ by (27);
        multiply to obtain $M'_{j,i}$ by (23).
      Orthogonalize $M'_{j,i}$ to obtain $U_{j,i}$ by (21).
   enddo
enddo
```

PROCEDURE TO COMPUTE UTU^T
Comment [Input to this procedure consists of n, k, the matrices $U_{j,i}$ computed above, a function to compute elements of T, and the chosen precision ϵ. Output is a matrix R_l such that $\|R_l - UTU^T\| < \epsilon \|T\|$.]

Step 4

Compute the $k \times k$ extracts, indicated by (28), of the submatrices of T shown in Fig. 4.

Step 5

Extract the matrices P'' (29) from $U_1, U_2U_1, \ldots, U_{l-2} \cdots U_1$ and compute $W_0, \ldots, W_{l-2}$ according to (30).

Step 6

Compute $R_0, \ldots, R_l$ by (31), discarding elements below a threshold τ determined by the precision ϵ (32).

PROCEDURE TO COMPUTE $UT^{-1}U^T$
Comment [Input to this procedure consists of n, the matrix R_l which approximates UTU^T, and the precision ϵ. Output is a matrix X_m which approximates $UT^{-1}U^T$.]

Step 7

Compute the matrix $X_0 = R_l{}^T / \|R_l{}^T R_l\|$ by direct matrix multiplication, discarding elements below a threshold τ determined by the precision ϵ (32).

Step 8

Comment [Obtain the inverse by Schulz iteration.]

```
do $m = 0, 1, \ldots$ while $\|I - X_m R_l\| \geq \epsilon$
   Compute $X_{m+1} = 2X_m - X_m R_l X_m$, discarding elements below threshold.
enddo
```

3.4. Complexity analysis. In Table 1, we provide the operation count for each step of the computation of $UT^{-1}U^T$.

TABLE 1

Step	Complexity	Explanation
1	$O(nk)$	There are $n/(2k)$ $2k \times 2k$ matrices; each element of the matrices is computed in constant time.
2	$O(nk^2)$	For each of the $n/(2k)$ matrices, perform a Gram–Schmidt orthogonalization requiring order $O(k^3)$ operations.
3	$O(nk^2)$	For each of $n/(4k)+n/(8k)+\cdots+1 = n/(2k)-1$ matrices, compute four products of a $k \times 2k$ matrix with a $2k \times 2k$ matrix, construct two $2k \times 2k$ matrices, and orthogonalize one $2k \times 2k$ matrix.
4	$O(nk)$	There are $6(1+3+7+\cdots+(n/(2k)-1))+ 3(n/k)-2$, or order $O(n/k)$, submatrices of T and for each matrix we compute k^2 elements.
5	$O(nk^2)$	There are $n/(2k)+n/(4k)+\cdots+1 = n/k-1$ matrices P'', each the product of two $k \times k$ matrices. These are each inverted and multiplied with the $O(n/k)$ matrices of the previous step.
6	$O(n \log n)$	The diagonally banded matrix W_0, which contains $O(n)$ elements, grows to $O(n \log n)$ elements by the computation of UW_0U^T, as can be seen by simply examining pictures of W_0 and U. The nonzero elements of the transformed $W_1, \ldots, W_{l-2}$ are a subset of those of W_0.
7	$O(n \log^2 n)$	Multiplication of two matrices, each with order $O(n \log n)$ elements, to obtain a product with order $O(n \log n)$ elements.
8	$O(n \log^2 n)$	Two multiplications like that of Step 7 are made per iteration; the number of iterations is independent of n and given by bound (14).
Total	$O(n \log^2 n)$	

4. Numerical examples.

In this section we present operators from several integral equations, the discretization and transformation of the operators to our wavelet-like bases, and the inversion of the operators via Schulz method.

4.1. Uncorrected quadratures. We first examine simple quadratures with equal weights, except weight zero at the singularity, as represented by matrix $T = T(n)$ defined by (12). We transform the matrix $I - T$ to wavelet-like coordinates as described in §3.2, then compute $(I - T)^{-1}$.

These discretizations are not particularly useful for the solution of the integral equations, due to their slow convergence to the integral operators. They nonetheless make good illustrative examples, for they retain the smoothness of the operator kernels and produce correspondingly sparse matrices. In the next section, we examine the results of using high-order quadratures.

For various sizes n of discretization, we tabulate the average number of elements per row in the transformed matrix $U(I - T)U^T$ and the computation time to obtain the matrix. In addition, we display the average number of elements per row of its inverse, and the time to compute the inverse. Finally, we show the error introduced by these computations. The error is determined by the application of the forward and inverse transformations to a random vector: Choose a vector v of length n with uniformly distributed pseudorandom elements; compute $(I - T)v$ directly, by a standard procedure requiring order $O(n^2)$ operations; transform to wavelet-like coordinates, obtaining $U(I - T)v$; apply the computed value of $U(I - T)^{-1}U^T$ to the vector $U(I - T)v$; transform to original coordinates by application of U^T; compare the result v' to v. The measure of error is the relative L^2 error, defined by the formula

$$e_{L^2} = \left(\frac{\sum_{i=1}^{n} [v_i' - v_i]^2}{\sum_{i=1}^{n} v_i{}^2} \right)^{1/2}.$$

The programs to transform and invert, as well as those to determine the error, were implemented in FORTRAN. All computations were performed in double-precision arithmetic on a Sun Sparcstation 1.

The first set of examples is for the kernel $K(x,t) = \log|x - t|$, for a wavelet-like basis of order $k = 4$ and various choices of precision ϵ. The matrix sparsities, execution times, and errors appear in Table 2. Although the sparse matrices are not banded, we loosely refer to the average number of matrix elements per row as the matrix *bandwidth*. We make the following observations.

TABLE 2
The operator $I - \mathcal{K}$ defined by the formula $((I - \mathcal{K})f)(x) = f(x) - \int_0^1 \log|x - t|\, f(t)\, dt$ is discretized, transformed to the wavelet-like coordinates with $k = 4$, and inverted. For various precisions ϵ and various sizes of discretization, we tabulate the average number of elements/row N_1 of the matrix in wavelet-like coordinates and the time in seconds t_1 to compute it, corresponding statistics N_2 and t_2 for the inverse, and the error (see text).

		Transform.		Inversion		L^2
ϵ	n	N_1	t_1	N_2	t_2	Error
10^{-2}	64	7.2	2	8.3	2	0.503E−02
	128	5.9	3	6.5	4	0.257E−02
	256	3.8	7	4.4	4	0.250E−02
	512	2.8	13	3.1	6	0.236E−02
	1024	1.9	26	2.1	6	0.227E−02
	2048	1.4	49	1.4	6	0.221E−02
	4096	1.2	97	1.2	8	0.221E−02
	8192	1.1	195	1.1	12	0.217E−02
10^{-3}	64	17.6	2	19.5	14	0.350E−03
	128	18.1	5	20.0	36	0.270E−03
	256	18.0	11	20.0	83	0.331E−03
	512	14.5	21	15.7	123	0.257E−03
	1024	13.3	41	15.5	262	0.340E−03
	2048	8.5	73	9.8	287	0.233E−03
	4096	5.8	131	6.5	304	0.222E−03
	8192	3.7	242	4.4	312	0.221E−03
10^{-4}	64	28.4	3	30.3	36	0.104E−03
	128	32.1	6	34.3	111	0.140E−03
	256	34.5	15	37.5	302	0.161E−03
	512	33.1	31	35.8	618	0.177E−03
	1024	30.2	63	33.6	1280	0.189E−03
	2048	25.0	121	27.6	2040	0.192E−03

1. The bandwidths N_1, N_2 of the operator and its inverse *decrease* with increasing matrix size. In other words, in the range of matrix sizes tabulated, the number of matrix elements grows more slowly than the matrix dimension n.

2. The operator matrix in wavelet-like coordinates is computed in time that grows nearly linearly in n.

3. The inverse matrix is computed in time which grows sublinearly in n. This is due to the fact that the cost of multiplying the sparse matrices is roughly order $O(nN^2)$, for size n and bandwidth N. One result is that the cost sometimes drops as n increases.

4. The accuracy is within the precision specified. In fact, due to the conservative element thresholding (32), the actual error is considerably less than ϵ.

5. The cost increases with increasing precision ϵ, due to the increasing bandwidths generated. The bandwidths increase approximately as $\log(1/\epsilon)$.

6. For $k = 4$, our fast transformation algorithm does not maintain the specified precision of $\epsilon = 10^{-4}$. This anticipated result follows from the error estimate for polynomial interpolation of logarithm on intervals separated from the origin. An unanticipated attendant result is that the bandwidth increases as the quality of approximation deteriorates (compare to $k = 8$, below). As a result, we did not complete examples for $n = 4096$, 8192.

7. The inversion of the 8192×8192 matrix preserving three-digit accuracy is done in five minutes on the Sparcstation. This compares to 95 days (estimated) for inverting the dense matrix by Gauss–Jordan and to 24 minutes for one dense matrix–vector multiplication of that size.

The condition number of the problem, as approximated by the product of the row-sum norms of $U(I - T)U^T$ and its computed inverse, is 3 (independent of size). Five iterations were required by the Schulz method to achieve convergence.

In Fig. 5 we show stages in the transformation of the matrix $I - T$. In particular, for $\epsilon = 10^{-3}$ and $n = 64$, the matrices $R_0, \ldots, R_{l-1}$ defined in (31) are shown. In addition, for $n = 128$, the transformed matrix $U(I - T)U^T$ and its inverse are shown in Fig. 6.

In the next set of examples, for which results are displayed in Table 3, we used the wavelet-like basis of order $k = 8$. We observe the following.

1. The bandwidths of the operator matrix and its inverse are less for $k = 8$ than for $k = 4$. The inversion times are correspondingly smaller.

2. The time required to compute the operator matrix is almost four times as large as that for $k = 4$. This is due to the cost of transforming the near-diagonal band, which is twice as wide for $k = 8$ as for $k = 4$.

3. The obtained accuracy exceeds the specified precision consistently.

4. As for $k = 4$, the scaling with size n is linear for the transformation step and sublinear for the inversion step.

In the final set of examples in which uncorrected quadratures were used, we perform computations for $k = 4$ and $\epsilon = 10^{-3}$, with various operator kernels. Table 4 presents the results. The first three kernels contain singularities of the types $s(x) = \log(x)$ and $s(x) = x^\alpha$ for $\alpha = \pm\frac{1}{2}$, and are nonsymmetric and nonconvolutional. It is readily seen that the bandwidth is strongly dependent on the type of singularity, with the singularity $x^{-1/2}$ producing the greatest bandwidth. We mention also that this particular integral equation is poorly conditioned; the condition numbers of the discretizations for $n = 64, 128, 256, 512, 1024$ are $9, 17, 34, 98, 469$, respectively.

The fourth kernel provides an example with an oscillatory coefficient $p(x) = (1 + \frac{1}{2}\sin(100x))$. The bases developed in §2.4, which depend on p, are used to transform the discretized integral operator to sparse form. We see in Table 4 that the inverse is also very sparse.

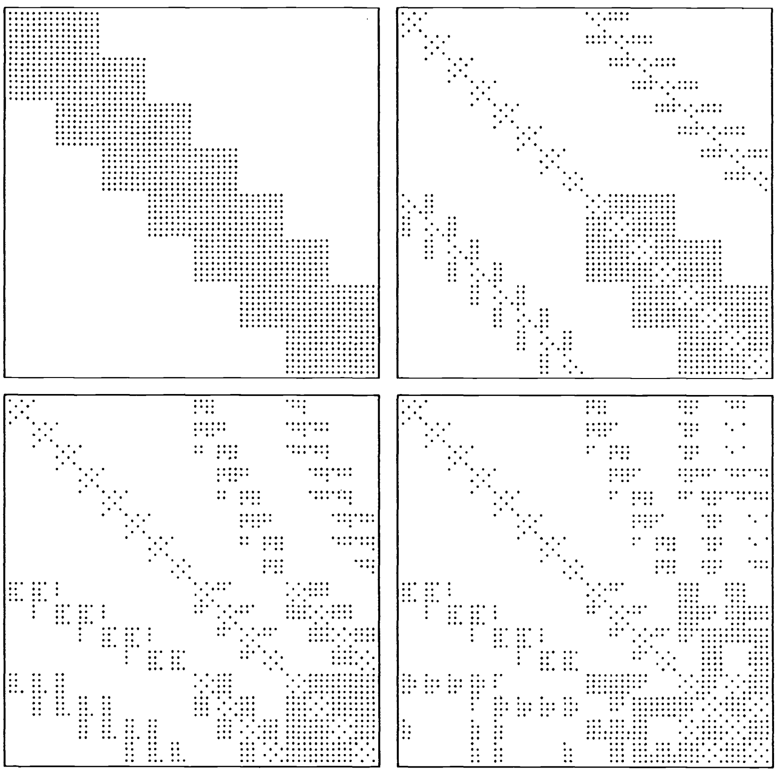


FIG. 5. *The matrices constructed in the transformation of* $I - T$*, matrices* $R_0, \ldots, R_3$ *defined in* (31)*, are shown for kernel* $K(x,t) = \log|x-t|$, $\epsilon = 10^{-3}$*, and* $n = 64$*. Matrix* R_4 *looks like* R_3 *and is not shown.*

4.2. Solution of integral equations. In the preceding subsection, we examined the characteristics of various integral operators and their inverses in wavelet-like coordinates. We used completely straightforward discretizations; the quadratures represented sums of the integrands at equispaced points (excluding singular points). Such simple quadratures converge too slowly to the integral operators to be of much use in solving integral equations, and we now turn to the high-order quadratures developed in [3].

We first present examples that correspond to the various kernels already tested and shown in Table 4. In Table 5 we tabulate the results, and bandwidth differences from Table 4 reflect the effect of the quadratures.

For the remaining examples we choose integral equations that can be solved analytically, so that the accuracy of the method can be checked. We consider a class of integral equations with logarithmic kernel,

$$f(x) - p(x) \int_0^1 \log|x-t|\, f(t)\, dt = g_m(x), \qquad x \in [0,1], \tag{33}$$

where the right-hand side g_m is chosen so that the solution f is given by the formula $f(x) = \sin(mx)$. The integration can be performed explicitly, yielding

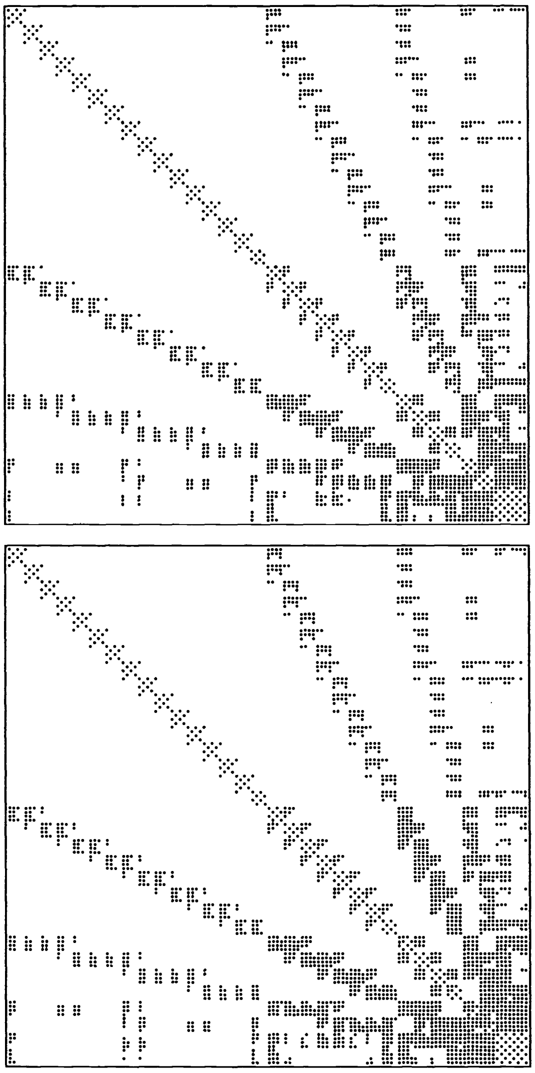


FIG. 6. *Transformed matrix $U(I-T)U^T$ (top) and its inverse (bottom) are shown for kernel $K(x,t)=\log|x-t|$, $\epsilon=10^{-3}$, and $n=128$.*

$$\int_0^1 \log|x-t|\, m\,\sin(mt)\,dt = \log(x)-\cos(m)\log(1-x) \\ -\cos(mx)[\mathrm{Ci}(mx)-\mathrm{Ci}(m(1-x))] \\ -\sin(mx)[\mathrm{Si}(mx)+\mathrm{Si}(m(1-x))],$$

where Ci and Si are the cosine integral and sine integral (see, e.g., [1, p. 231]). Equation (33) clearly requires quadratures with increasing resolution as m increases; for our examples we let $n=m$, which corresponds to 2π points per oscillation of the right-hand side g_m.

TABLE 3

The operator $I - \mathcal{K}$ defined by the formula $((I - \mathcal{K})f)(x) = f(x) - \int_0^1 \log|x-t|\, f(t)\,dt$ is discretized, transformed to the wavelet-like coordinates with $k = 8$, and inverted. (See Table 2 and text.)

		Transform.		Inversion		L^2
ϵ	n	N_1	t_1	N_2	t_2	Error
10^{-2}	64	5.8	4	6.2	1	0.191E−02
	128	5.0	10	5.5	2	0.368E−02
	256	3.3	22	3.6	3	0.184E−02
	512	2.7	46	2.9	4	0.113E−02
	1024	1.8	92	1.8	4	0.177E−02
	2048	1.4	182	1.4	5	0.170E−02
	4096	1.2	363	1.2	8	0.928E−03
	8192	1.1	729	1.1	11	0.166E−02
10^{-3}	64	13.4	5	14.5	8	0.373E−03
	128	14.2	13	15.5	21	0.332E−03
	256	13.5	28	14.5	46	0.259E−03
	512	12.7	57	13.6	90	0.225E−03
	1024	10.2	114	11.1	134	0.198E−03
	2048	7.7	221	8.3	176	0.179E−03
	4096	4.9	429	5.2	185	0.174E−03
	8192	3.5	818	3.7	208	0.173E−03
10^{-4}	64	21.8	6	23.0	23	0.280E−04
	128	26.3	15	28.0	81	0.253E−04
	256	28.7	35	31.0	235	0.246E−04
	512	28.4	75	30.9	538	0.184E−04
	1024	25.5	149	27.2	969	0.925E−05
	2048	22.0	297	23.8	1739	0.899E−05
	4096	17.7	561	19.1	2610	0.798E−05

Initially we choose coefficient $p(x) = 1$. The results are given in Table 6. Here the error shown is the error of the computed solution relative to the true solution of the integral equation. Many of the observations of the preceding examples can be repeated here; additionally, we make the following comments.

1. The bandwidths are greater than for the uncorrected quadratures, but this effect generally decreases with increasing size.

2. The integral equations are solved to within the specified precision in every case but one. The exception, for $\epsilon = 10^{-4}$ and $n = 64$, is likely due to the small number of quadrature points and high specified precision.

3. An integral equation requiring an 8192-point discretization is solved to three-digit accuracy in less than 20 minutes on the Sparcstation.

For our second set of integral equations, we let the coefficient p be the oscillatory function given by the formula $p(x) = 1 + \frac{1}{2}\sin(100x)$. We carry out the transformation described in §2.4 to solve the integral equation (33). The results are shown in Table 7, and as with Table 6, the error refers to the error of the computed solution relative to the true solution of the integral equation. For the oscillatory coefficient we see performance similar to the constant-coefficient problem, but the cost is higher.

5. Generalizations and applications. In this paper, we have constructed a new class of vector-space wavelet-like bases in which a variety of integral operators

TABLE 4

The operator $I-\mathcal{K}$ defined by the formula $((I-\mathcal{K})f)(x) = f(x) - \int_0^1 K(x,t)\, f(t)\, dt$, for nonsymmetric, nonconvolutional kernels $K(x,t)$ shown below, is discretized, transformed to the wavelet-like coordinates with $k = 4$ and $\epsilon = 10^{-3}$, and inverted. (See Table 2 and text.)

		Transform.		Inversion		L^2
$K(x,t)$	n	N_1	t_1	N_2	t_2	Error
$\cos(xt^2)\log\lvert x-t\rvert$	64	18.2	2	20.2	15	0.318E−03
	128	18.6	5	20.4	37	0.302E−03
	256	17.9	11	19.8	82	0.301E−03
	512	14.9	22	16.3	131	0.284E−03
	1024	12.9	42	14.7	242	0.315E−03
	2048	8.5	76	9.5	283	0.241E−03
	4096	5.5	137	6.1	291	0.231E−03
	8192	3.6	252	4.3	310	0.230E−03
$\cos(xt^2)\lvert x-t\rvert^{-1/2}$	64	27.2	3	28.9	32	0.256E−03
	128	31.6	7	34.1	122	0.357E−03
	256	35.6	16	40.6	454	0.434E−03
	512	37.3	35	46.3	1509	0.643E−03
	1024	34.5	72	45.4	4166	0.821E−03
$\cos(xt^2)\lvert x-t\rvert^{1/2}$	64	6.8	2	7.3	2	0.303E−03
	128	4.4	4	4.7	2	0.204E−03
	256	2.9	8	3.0	3	0.209E−03
	512	2.1	15	2.3	3	0.165E−03
	1024	1.5	30	1.5	3	0.208E−03
	2048	1.4	60	1.4	6	0.909E−03
	4096	1.1	119	1.2	7	0.614E−03
	8192	1.1	242	1.1	12	0.666E−03
$(1+\frac{1}{2}\sin(100x))\times$	64	30.5	3	33.8	44	0.344E−03
$\log\lvert x-t\rvert$	128	31.8	6	35.1	103	0.363E−03
	256	21.2	12	24.1	119	0.348E−03
	512	18.6	23	20.7	225	0.372E−03
	1024	15.8	45	18.4	404	0.392E−03
	2048	10.6	82	12.2	466	0.355E−03
	4096	6.4	145	7.4	497	0.336E−03
	8192	4.0	265	4.6	510	0.331E−03

are represented as sparse matrices. The inverses of these matrices are also sparse, a fact which enables the corresponding integral equations to be solved rapidly. We have asserted that the time complexity for an n-point discretization is bounded by order $O(n \log^2 n)$, but observed order $O(n)$ performance in practice. This cost should be contrasted with a cost of order $O(n^2)$ for direct application of a dense matrix, and order $O(n^3)$ for direct inversion.

A number of limitations exist in the procedures described above. These restrictions may be categorized as "software limitations" and "research questions." We discuss software limitations first.

5.1. Software limitations. Throughout the paper, we have assumed that the size of the problem n has the form $n = 2^l k$ for some l. This restriction is not fundamental; it merely simplifies the software.

A second software restriction is the assumption of only diagonal singularities. This case is an important one in practice, but in certain situations we may encounter

TABLE 5

The operator $I-\mathcal{K}$ defined by the formula $((I-\mathcal{K})f)(x) = f(x)-\int_0^1 K(x,t)\,f(t)\,dt$, for nonsymmetric, nonconvolutional kernels $K(x,t)$ shown below, is discretized with the corrected trapezoidal rules, transformed to the wavelet-like coordinates with $k = 4$ and $\epsilon = 10^{-3}$, and inverted. (Compare to Table 4.)

		Transform.		Inversion		L^2
$K(x,t)$	n	N_1	t_1	N_2	t_2	Error
$\cos(xt^2)\log\lvert x-t\rvert$	64	28.3	4	31.6	38	0.164E−03
	128	31.5	9	34.3	103	0.162E−03
	256	30.8	21	33.9	221	0.172E−03
	512	27.0	41	29.7	370	0.177E−03
	1024	21.0	80	23.7	454	0.357E−03
	2048	14.8	143	17.2	566	0.317E−03
	4096	9.5	250	10.4	555	0.282E−03
	8192	5.8	448	6.9	665	0.271E−03
$\cos(xt^2)\lvert x-t\rvert^{-1/2}$	64	32.4	4	39.8	87	0.133E−02
	128	38.3	10	45.7	251	0.412E−03
	256	42.7	23	49.3	638	0.464E−03
	512	45.1	51	51.3	1494	0.562E−03
	1024	46.2	110	52.1	3309	0.635E−03
$\cos(xt^2)\lvert x-t\rvert^{1/2}$	64	10.4	3	18.4	9	0.867E−03
	128	7.6	6	13.8	13	0.526E−03
	256	5.1	13	9.3	16	0.358E−03
	512	3.3	25	5.2	15	0.292E−03
	1024	2.3	48	3.1	15	0.201E−03
	2048	1.9	96	2.3	20	0.393E−03
	4096	1.5	188	1.7	25	0.405E−03
	8192	1.3	374	1.4	36	0.404E−03

singularities or near-singularities off the main diagonal. The scheme described in §3.2 for transformation of a matrix to wavelet-like bases can be readily revised to an adaptive scheme, which works as follows: an $m \times m$ submatrix A is transformed to wavelet-like coordinates under the assumption that it can be approximated to high precision along both rows and columns by polynomials of degree less than k. This assumption is then checked by dividing A into four submatrices, each of dimension $m/2 \times m/2$, transforming each submatrix, and "gluing" the pieces together. If the results from the two computations match (to high precision), no further refinement of the original submatrix is needed. Otherwise, the procedure is repeated recursively on the $m/2 \times m/2$ submatrices. The cost of this adaptive procedure is roughly five times as great as the cost of a static procedure in which the structure of the singularities is known a priori.

5.2. Research questions. The list of research issues is, of course, much longer. One of the most pressing issues is the generalization to two and three dimensions. Although, conceptually, the generalization of the wavelet-like bases to several dimensions is quite straightforward (see, e.g., [2]), actual procedures to perform the required orthogonalizations have not been developed. Also, the issue of high-order quadratures for two and three dimensions has not been resolved.

Another question is whether similar "custom-constructed" bases can be used to create sparse representations of integral operators with oscillatory kernels. Initial

TABLE 6

The integral equations $f(x) - \int_0^1 \log|x-t|\, f(t)\, dt = g_m(x)$, for which an explicit solution is known, are solved by the methods of this chapter (compare to Table 2 and see text). For $\epsilon = 10^{-2}, 10^{-3}, 10^{-4}$ we set $k = 4, 4, 8$, respectively.

		Transform.		Inversion		L^2
ϵ	n, m	N_1	t_1	N_2	t_2	Error
10^{-2}	64	11.4	3	14.4	7	0.283E−02
	128	10.7	7	13.2	14	0.212E−02
	256	8.6	13	10.6	20	0.140E−02
	512	6.3	26	7.6	26	0.112E−02
	1024	3.6	48	4.5	28	0.821E−03
	2048	1.9	90	2.3	21	0.932E−03
	4096	1.3	174	1.5	15	0.674E−03
	8192	1.1	344	1.1	13	0.499E−03
10^{-3}	64	27.7	4	31.3	36	0.235E−03
	128	31.0	9	34.2	99	0.169E−03
	256	30.6	20	33.6	215	0.161E−03
	512	27.5	41	30.2	377	0.130E−03
	1024	21.7	79	24.4	470	0.597E−03
	2048	15.5	143	18.1	604	0.479E−03
	4096	9.7	248	10.6	579	0.415E−03
	8192	6.0	444	7.3	690	0.354E−03
10^{-4}	64	37.2	8	45.9	78	0.127E−03
	128	47.1	23	56.5	278	0.473E−04
	256	52.9	54	60.9	745	0.311E−04
	512	55.0	118	61.4	1701	0.100E−04
	1024	52.3	248	57.2	3287	0.734E−05

efforts in this direction for a limited class of such operators, in particular for Fourier transforms with nonequispaced points and frequencies, appear promising [9].

5.3. Applications. In this paper the primary application of our new wavelet-like bases has been the solution of second-kind integral equations. The bases are very effective for the fast solution of a wide class of such problems. In addition, we expect many other classes of problems to be solved efficiently using these techniques. We list a few of these problem types.

1. Elliptic partial differential equations rewritten as integral equations by the Lippman–Schwinger method, in which the Green's functions are nonoscillatory.

2. Evolution of homogeneous parabolic partial differential equations (PDEs) with constant or periodic boundary conditions, by explicit time steps. This method consists of repeated squarings of the operator for a single time step, leading to an order $O(n \log t)$ algorithm for evolving an n-point discretization for t time steps.

3. Evolution of general parabolic PDEs by implicit time steps, in which the elliptic problem on each time step is solved in wavelet-like coordinates.

4. Evolution of hyperbolic PDEs by a method of operator squaring analogous to the scheme proposed for homogeneous parabolic PDEs above.

5. Problems of potential theory and pseudodifferential operators.

6. Signal compression, including signals of seismic, visual, and vocal origin. There is also reason to expect that analysis of such compressed data will be simpler than analysis of data resulting from less efficient compression schemes.

TABLE 7
The integral equations $f(x) - p(x)\int_0^1 \log|x-t|\,f(t)\,dt = g_m(x)$, for which an explicit solution is known, are solved by the methods of this chapter (compare to Table 2 and see text). For $\epsilon = 10^{-2}, 10^{-3}, 10^{-4}$ we set $k = 4, 4, 8$, respectively.

		Transform.		Inversion		L^2
ϵ	n, m	N_1	t_1	N_2	t_2	Error
10^{-2}	64	19.7	4	23.9	18	0.360E−02
	128	17.7	8	21.0	36	0.182E−02
	256	12.6	15	14.6	47	0.174E−02
	512	8.4	29	9.8	57	0.112E−02
	1024	4.7	55	5.7	56	0.104E−02
	2048	2.4	103	2.7	45	0.902E−03
	4096	1.6	198	1.7	38	0.720E−03
	8192	1.3	392	1.3	35	0.543E−03
10^{-3}	64	36.2	4	41.3	63	0.228E−02
	128	40.8	10	47.0	186	0.209E−03
	256	40.5	23	47.3	427	0.177E−03
	512	34.7	46	40.9	712	0.125E−03
	1024	26.6	87	32.5	1042	0.134E−03
	2048	18.7	158	22.5	1065	0.597E−03
	4096	12.2	281	14.2	1127	0.529E−03
	8192	7.2	502	8.4	1104	0.461E−03
10^{-4}	64	47.6	9	58.2	123	0.230E−02
	128	60.7	25	77.3	479	0.180E−03
	256	64.1	59	81.2	1204	0.124E−03
	512	62.5	128	76.3	2492	0.125E−04
	1024	58.8	267	69.3	4672	0.862E−05

In this paper we strayed from the original mathematical definition of wavelets to construct classes of bases tailored for numerical computation. The basis vectors' principal properties of local support and vanishing moments lead to sparse representations of functions and operators that are smooth except at a small number of singularities. There is little doubt that other bases can be constructed along similar lines to possess various properties. One current challenge is the construction of bases suitable for the efficient representation of a variety of oscillatory operators.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, D.C., 1972.

[2] B. Alpert, *A class of bases in L^2 for the sparse representation of integral operators*, Tech. Rep., Lawrence Berkeley Laboratory, University of California, Berkeley, CA, 1990; SIAM J. Math. Anal., 13(1993), to appear.

[3] ———, *Rapidly-convergent quadratures for integral operators with singular kernels*, Tech. Rep., Lawrence Berkeley Laboratory, University of California, Berkeley, CA, 1990.

[4] ———, *Sparse representation of smooth linear operators*, Ph.D. thesis, Dept. of Computer Science, Yale University, New Haven, CT, Dec. 1990.

[5] B. Alpert and V. Rokhlin, *A fast algorithm for the evaluation of Legendre expansions*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 158–179.

[6] G. Beylkin, R. Coifman, and V. Rokhlin, *Fast wavelet transforms and numerical algorithms* I, Comm. Pure Appl. Math., XLIV (1991), pp. 141–183.

[7] I. DAUBECHIES, *Orthonormal bases of compactly supported wavelets*, Comm. Pure Appl. Math., XLI (1988), pp. 909–996.

[8] L. M. DELVES AND J. L. MOHAMED, *Computational Methods for Integral Equations*, Cambridge University Press, London, 1985.

[9] A. DUTT AND V. ROKHLIN, Personal communication, 1990.

[10] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

[11] L. GREENGARD AND J. STRAIN, *The fast Gauss transform*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 79–84.

[12] A. GROSSMAN AND J. MORLET, *Decomposition of Hardy functions into square integrable wavelets of constant shape*, SIAM J. Math. Anal., 15 (1984), pp. 723–736.

[13] Y. MEYER, *Principe d'incertitude, bases Hilbertiennes et algèbres d'opérateurs*, Tech. Rep., Séminaire Bourbaki, nr. 662, 1985–1986.

[14] S. O'DONNELL AND V. ROKHLIN, *A fast algorithm for the numerical evaluation of conformal mappings*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 475–487.

[15] G. SCHULZ, *Iterative berechnung der reziproken matrix*, Z. Angew. Math. Mech., 13 (1933), pp. 57–59.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 185–217, January 1993

AN h-r-ADAPTIVE APPROXIMATE RIEMANN SOLVER FOR THE EULER EQUATIONS IN TWO DIMENSIONS*

MICHAEL G. EDWARDS†‡, J. TINSLEY ODEN†, AND LESZEK DEMKOWICZ†

Abstract. A new adaptive strategy for solving the Euler equations of compressible flow is presented. The method of Roe [*J. Comput. Phys.*, 43 (1981), pp. 357–372] is extended into two dimensions for an arbitrary quadrilateral grid and is coupled with the h-adaptive quadrilateral refinement-unrefinement algorithm of Demkowicz and Oden [*TICOM* Report 88-02]. Refinement of a quadrilateral grid retains a certain grid structure which is fully exploited by the extension of the higher-order version of the method into two dimensions. A total variation diminishing (TVD) analysis is presented for a nonuniform grid, together with an assessment of the solution error induced by the nonuniformity in the grid. Grid movement is also considered and adaptive strategies are discussed and tested. The adaptive scheme proves to be highly robust. Improved accuracy and large savings in computer time are obtained.

Key words. adaptive, grid refinement, higher-order, TVD, Riemann solver, compressible Euler equations

AMS(MOS) subject classifications. 65M06, 65M50

1. Introduction.

In the early seventies, much of the research on numerical methods for hyperbolic conservation laws focused on producing schemes that were known to produce physically meaningful solutions whenever they converged to the exact solution. At the same time, schemes were sought that did not oscillate in the vicinity of shocks. The first family of schemes that fulfilled these requirements were the so-called monotone difference schemes in which the numerical fluxes are monotone functions of the cell-centered values of the discrete solutions.

Unfortunately, monotone schemes were found to suffer from two major deficiencies: they are no more than first-order accurate, and they are usually overdissipative, smearing shocks over several grid spacings. These defects promoted an extensive series of investigations for the "holy grail" in conservation law solvers: schemes that did not oscillate but did yield higher-order (e.g., second-order) accuracy.

An advance in this direction for the case of one-dimensional scalar conservation laws was made by Harten [3], who introduced the notion of a TVD (total variation diminishing) scheme. The idea is that if the total variation of the solution can be controlled so that it never increases over a timestep, then a nonoscillating solution with second-order accuracy can be obtained. This can be accomplished by limiting the values of the numerical flux ("flux-limiting methods"); several alternative flux-limiting strategies were discussed by Sweby [4]. Among these is the method of Roe [1], which, while usually very effective, may violate the entropy condition for the conservation law. An "entropy fix" was proposed by Harten and Hyman [5] for overcoming this defect in Roe's approach.

It should also be noted that most of the theoretical results were developed for one-dimensional scalar conservation laws. Goodman and LeVeque [6] argued that

*Received by the editors June 6, 1989; accepted for publication (in revised form) March 31, 1992.

†Texas Institute for Computational Mechanics, The University of Texas at Austin, Austin, Texas 78712. The research of the first two authors was supported in part by Army Research Office grant DAAL03-89-K-0120 and by the Office of Naval Research. The research of the third author was supported by NASA Langley Research Center.

‡Present address, B. P. Research Centre, Chertsey Road, Sunbury-on-Thames, Middlesex TW16 7LN, United Kingdom.

any two-dimensional TVD scheme had to also be monotone, and hence, may be only first-order accurate. However, numerical experiments suggest that two-dimensional generalizations of one-dimensional TVD schemes (which are not strictly TVD schemes in two dimensions) can be second-order accurate in many cases. Moreover, they can also be robust and yield nonoscillating, high-resolution simulation of shocks (see, e.g., Yee [7]).

The simple but less reliable alternative to using a TVD scheme is to resort to some variant of a Lax–Wendroff scheme and combine it with an ad hoc artificial viscosity method, which can be tuned to eliminate spurious oscillations in some cases [8].

However, the resolution of discontinuities obtained by any discrete scheme will still be limited by the local cell size. Adaptive grid techniques have been developed by several authors, (e.g., [9], [10], [11], [12], and [13]) with a common aim of concentrating (either by inserting or moving) grid nodes into the areas of the flow field where they are most needed (where the flow gradients are large), thereby reducing the discretization error as the element size reduces. For a numerical scheme of given accuracy, grid adaption can lead to increased accuracy and a reduction in computer time and computer storage requirements.

Although adaptive techniques can improve on the results obtained with artificial viscosity methods, any spurious oscillations that occur can be highlighted by the adaptive grid and amplified [9]. This provides a strong motivation for combining TVD schemes with adaptivity. In particular, the combination of a second-order flux-limited scheme on an adaptive quadrilateral grid has a two-fold advantage over its triangular counterpart.

First, a dynamically refined quadrilateral grid is far less likely to produce badly distorted elements than a dynamically refined triangular grid, and therefore, we can expect convergence on a quadrilateral grid.

Second, the flux-limiting concept can be naturally generalized to an adaptive quadrilateral grid.

In this paper, we develop an adaptive, two-dimensional TVD scheme for systems of hyperbolic conservation laws, particularly Euler's equations. This scheme functions on an arbitrary unstructured mesh of quadrilateral cells (unstructured in the sense that the cells may be dynamically refined or unrefined). We employ an extended version of Roe's scheme in two dimensions with flux limiting and an entropy fix.

Error indicators are computed at the end of a designated number of timesteps, and the mesh is automatically refined (an h-method) or unrefined using the technique of Demkowicz and Oden [2]. This adaptive technique is applicable to both time-dependent and steady-state problems.

In addition (currently only) for steady-state problems, grid node relocation (an r-method) is induced according to an equidistribution principle. The grid is postprocessed and automatically aligns grid lines along discontinuities.

The r-method is also combined with the h-method (currently) for steady-state problems. After the grid has been relocated and subsequently refined, results superior to those of h-adaptive schemes without node relocation are obtained.

In all of the numerical experiments performed, the adaptive h-scheme (for time-dependent and steady-state problems) and the h-r scheme (for steady-state problems) perform surprisingly well, giving excellent resolution of flow features on rather coarse grids. Large savings in computer time were also observed with the same code taking four times longer to run on equivalent uniform fine grids.

2. Roe scheme. We begin by reviewing the scheme proposed by Roe [14], [15] for the scalar conservation law

$$\begin{aligned} &u_t + f(u)_x = 0 \quad t > 0 \quad x \in \mathbb{R}, \\ &u(x,0) = u_o(x), \end{aligned} \tag{2.1}$$

where $u(x,t)$ has initial data $u(x,0) = u_o(x)$. We also write (2.1) in the form

$$u_t + a(u)u_x = 0, \qquad a(u) = \frac{df}{du}.$$

We next partition the domain $\mathbb{R}$ into finite difference cells $I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ $i \in Z$, with centroids x_i. A piecewise constant mesh function W takes on values $W_i = W(x_i)$. We use the following standard notation for differences:

$$\Delta W_{i+\frac{1}{2}} = \Delta_- W_{i+1} = \Delta_+ W_i = W_{i+1} - W_i;$$

for the grid ratio μ:

$$\mu = \Delta t / \Delta x; \tag{2.2a}$$

for the local CFL number ν:

$$\nu_{i+\frac{1}{2}} = \frac{\mu \Delta f_{i+\frac{1}{2}}}{\Delta u_{i+\frac{1}{2}}} = \mu a(u_i, u_{i+1}) = \mu a_{i+\frac{1}{2}}; \tag{2.2b}$$

and $a(u_i, u_{i+1})$ is the discrete wave speed at $i + \frac{1}{2}$.

The first-order Roe [14] scheme (in space and time) is the conservative upwind scheme

$$u_i^{n+1} = \begin{cases} u_i^n - \nu_{i-\frac{1}{2}} \Delta u_{i-\frac{1}{2}}^n, & \nu_{i-\frac{1}{2}} \geq 0, \\ u_i^n - \nu_{i+\frac{1}{2}} \Delta u_{i+\frac{1}{2}}^n, & \nu_{i+\frac{1}{2}} < 0, \end{cases} \tag{2.3}$$

which is stable and oscillation free for

$$|\nu| \leq 1. \tag{2.4}$$

We note that (2.3) can be written as

$$u_i^{n+1} = u_i^n - \mu (h_{i+\frac{1}{2}} - h_{i-\frac{1}{2}}), \tag{2.5a}$$

where

$$h_{i+\frac{1}{2}} = h(u_{i+1}, u_i) = \tfrac{1}{2}(f(u_i^n) + f(u_{i+1}^n) - |a_{i+\frac{1}{2}}| \Delta u_{i+\frac{1}{2}}^n) \tag{2.5b}$$

is a consistent numerical flux function with

$$h(u,u) = f(u).$$

It is well known that this scheme must be modified to ensure entropy satisfaction, which is achieved here by employing the Harten and Hyman entropy fix [5]. Roe [14] extended his scheme to second-order accuracy and assured monotonicity preservation (monotone data at time level n remains monotone at time level $n+1$) by the use of a flux limiter, a device previously used by van Leer [16]. The monotonicity-preserving schemes of Chakravarthy and Osher [17] and Harten [3] employ similar devices, and the relationships between the schemes is discussed by Sweby [4], who presents a unified approach to designing such schemes.

All of the above-mentioned schemes can be shown to be TVD, a notion first introduced by Harten [3]. The total variation, at time level $n+1$, TV(u^{n+1}), of the solution is defined by

$$\text{TV}\ (u^{n+1}) = \sum_i \left|u^{n+1}_{i+1} - u^{n+1}_i\right|. \tag{2.6}$$

Harten [5] defined a TVD scheme for which

$$\text{TV}\ (u^{n+1}) \leq \ \text{TV}\ (u^n). \tag{2.7}$$

One immediate consequence of (2.7) is that the total variation remains bounded, which is one of the criteria that must be satisfied in order to establish convergence of a difference scheme approximating (2.1) (see [18], [19], and [20]).

An important practical consequence of (2.7) is that the solutions obtained from a TVD scheme are free of spurious oscillations, which follows from Harten's [3] observation that a TVD scheme is monotonicity preserving.

Harten [3] has also shown that sufficient conditions for a scheme of the form

$$u^{n+1}_i = u^n_i - C_{i-\frac{1}{2}}\Delta u^n_{i-\frac{1}{2}} + D_{i+\frac{1}{2}}\Delta u^n_{i+\frac{1}{2}} \tag{2.8a}$$

to be TVD are

$$0 \leq C_{i-\frac{1}{2}}, \quad 0 \leq D_{i+\frac{1}{2}}, \quad 0 \leq C_{i+\frac{1}{2}} + D_{i+\frac{1}{2}} \leq 1. \tag{2.8b}$$

From (2.3) and (2.4) , it follows that the first-order upwind scheme satisfies (2.8b) (and is therefore TVD) if for all i,

$$|\nu_{i+\frac{1}{2}}| \leq 1. \tag{2.9}$$

3. A TVD scheme on a nonuniform grid. While much attention has been focused on the application of the Roe scheme via the use of uniform grids, very little attention has been given in the case of nonuniform grids [21]. Since the usual definitions of accuracy on a uniform grid do not necessarily apply to nonuniform grids, we shall refer to the usual first-order scheme as the low-order scheme, and the usual second-order schemes as the high-order scheme on nonuniform grids.

In this section we shall consider how to construct a high-order TVD scheme on a nonuniform grid in one dimension. Our primary concern is that the scheme remains conservative on a nonuniform grid. We shall consider the question of accuracy after deriving a TVD scheme in conservation form on a nonuniform grid.

Consider the Lax–Wendroff scheme on a uniform grid:

$$u^{n+1}_i = u^n_i - \tfrac{1}{2}(\nu_{i-\frac{1}{2}}\Delta u^n_{i-\frac{1}{2}} + \nu_{i+\frac{1}{2}}\Delta u^n_{i+\frac{1}{2}}) + \tfrac{1}{2}(|\nu_{i+\frac{1}{2}}|^2\Delta u^n_{i+\frac{1}{2}} - |\nu_{i-\frac{1}{2}}|^2\Delta u^n_{i-\frac{1}{2}}). \tag{3.1}$$

This scheme may be written in conservation form as

$$u^{n+1}_i = u^n_i - \frac{\Delta t}{\Delta x}(h_{i+\frac{1}{2}} - h_{i-\frac{1}{2}}), \tag{3.2}$$

where

$$h_{i+\frac{1}{2}} = \left(\frac{f_i + f_{i+1}}{2}\right) - \frac{\Delta t}{2\Delta x}|a_{i+\frac{1}{2}}|^2 \Delta u^n_{i+\frac{1}{2}}. \tag{3.3}$$

The scheme (3.2) is said to be conservative since conservation may be demonstrated immediately by summing (3.2) over all grid points i to obtain

$$\sum_i (u_i^{n+1} - u_i^n)\Delta x = -\Delta t \sum_i (h_{i+\frac{1}{2}} - h_{i-\frac{1}{2}}). \tag{3.4}$$

Omitting boundary terms, the contribution on the right of (3.4) is zero, hence

$$\sum_i (u_i^{n+1} - u_i^n)\Delta x = 0, \tag{3.5}$$

and (3.5) indicates that the area (or "mass") $\int u dx$ is conserved. So far we have assumed that Δx is constant, but if we consider the nonuniform grid shown in Fig. 1, then we can replace Δx with the local element length Δx_i in (3.2)–(3.4) and retain conservation. The nonuniform grid approximation is

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x_i}(h_{i+\frac{1}{2}} - h_{i-\frac{1}{2}}), \tag{3.6}$$

where

$$h_{i+\frac{1}{2}} = \left(\frac{f_i + f_{i+1}}{2}\right) - \frac{\Delta t}{2\Delta x_{i+\frac{1}{2}}}|a_{i+\frac{1}{2}}|^2 \Delta u_{i+\frac{1}{2}}, \tag{3.7a}$$

$$\Delta x_{i+\frac{1}{2}} = (x_{i+1} - x_i). \tag{3.7b}$$

Note the introduction of $\Delta x_{i+\frac{1}{2}}$ in $h_{i+\frac{1}{2}}$, which ensures that the flux summation will continue to cancel for a nonuniform grid.

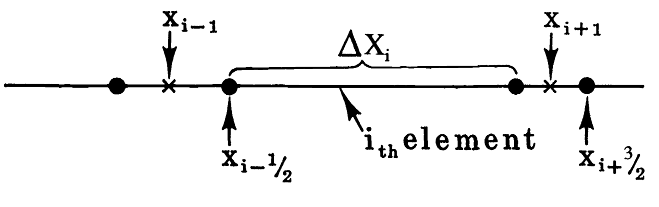


FIG. 1. *Nonuniform grid.*

The low-order scheme of §2 can be similarly generalized to a nonuniform grid with (3.7a) replaced by

$$h_{i+\frac{1}{2}} = \tfrac{1}{2}(f_i + f_{i+1}) - \tfrac{1}{2}|a_{i+\frac{1}{2}}|\Delta u_{i+\frac{1}{2}}, \tag{3.8}$$

while (3.6) remains unchanged. Expanding (3.6) and (3.8) and taking the case $a_{i+\frac{1}{2}} \geq 0$, the scheme can be written in the upwind form

$$u_i^{n+1} = u_i^n - \frac{\Delta t a_{i-\frac{1}{2}}}{\Delta x_i} \Delta u^n_{i-\frac{1}{2}}. \tag{3.9}$$

As in the case for uniform grids [4], the Lax–Wendroff scheme can be rearranged as the first-order upwind scheme together with an antidiffusive term, i.e.,

$$u_i^{n+1} = u_i^n - \frac{\Delta t a_{i-\frac{1}{2}}}{\Delta x_i} \Delta u_{i-\frac{1}{2}}^n - \frac{\Delta_-}{\Delta x_i} \left[\frac{\Delta t}{2} \left(1 - \nu_{i+\frac{1}{2}}\right) a_{i+\frac{1}{2}} \Delta u_{i+\frac{1}{2}}^n \right]. \tag{3.10}$$

Following the usual TVD analysis [4] an antidiffusive term can be added in the conservation form

$$\frac{\Delta_-}{\Delta x_i} \left(\frac{\Delta t}{2} \left(1 - \nu_{i+\frac{1}{2}}\right) a_{i+\frac{1}{2}} \left(1 - \phi_{i+\frac{1}{2}}\right) \Delta u_{i+\frac{1}{2}} \right), \tag{3.11}$$

where $\phi_{i+\frac{1}{2}}$ is the flux limiter. The resulting scheme is

$$u_i^{n+1} = u_i^n - \frac{\Delta t a_{i-\frac{1}{2}}}{\Delta x_i} \Delta u_{i-\frac{1}{2}}^n - \frac{\Delta_-}{\Delta x_i} \left(\frac{\Delta t}{2} \left(1 - \nu_{i+\frac{1}{2}}\right) a_{i+\frac{1}{2}} \phi_{i+\frac{1}{2}} \Delta u_{i+\frac{1}{2}}^n \right). \tag{3.12}$$

Rearranging (3.12) into the form

$$u_i^{n+1} = u_i^n - c_{i-\frac{1}{2}} \Delta u_{i-\frac{1}{2}}^n, \tag{3.13}$$

where

$$c_{i-\frac{1}{2}} = \frac{a_{i-\frac{1}{2}}}{\Delta x_i} \Delta t \left(1 - \left(\frac{\phi_{i+\frac{1}{2}}}{r_i^+} - \phi_{i-\frac{1}{2}} \right) \frac{(1 - \nu_{i-\frac{1}{2}})}{2} \right), \tag{3.14}$$

$$r_i^+ = \frac{a_{i-\frac{1}{2}}(1 - \nu_{i-\frac{1}{2}}) \Delta u_{i-\frac{1}{2}}}{a_{i+\frac{1}{2}}(1 - \nu_{i+\frac{1}{2}}) \Delta u_{i+\frac{1}{2}}}, \qquad \nu_{i+\frac{1}{2}} = \frac{a_{i+\frac{1}{2}} \Delta t}{\Delta x_{i+\frac{1}{2}}}, \tag{3.15}$$

and comparing with (2.8), it follows that the scheme will be TVD for

$$0 \le c_{i-\frac{1}{2}} \le 1. \tag{3.16}$$

Taking ϕ to be positive the left-hand inequality of (3.16) is satisfied for

$$\frac{\phi_{i+\frac{1}{2}}}{r_i^+} \le \frac{2}{(1 - \nu_{i-\frac{1}{2}})}. \tag{3.17}$$

The right-hand inequality is satisfied for

$$\phi_{i-\frac{1}{2}} \le \frac{2}{(1 - \nu_{i-\frac{1}{2}})} \left(\frac{1}{\nu_i} - 1 \right), \tag{3.18a}$$

where

$$\nu_i = \frac{a_{i-\frac{1}{2}} \Delta t}{\Delta x_i}. \tag{3.18b}$$

The bound on ϕ for a uniform grid

$$\phi \le \min(2 r_i^+, 2) \tag{3.19}$$

is certainly obtained if

$$\frac{1}{\max(\nu_i)} - 1 \geq 1, \tag{3.20}$$

and $0 \leq \nu_{i-\frac{1}{2}} \leq 1$, and (3.20) is satisfied if

$$a_{i-\frac{1}{2}}\Delta t \leq \frac{\Delta x_i}{2}, \tag{3.21}$$

which corresponds to a maximum Courant–Friedrichs–Lewy (CFL) condition of $\frac{1}{2}$, but with respect to the mean cell length. A similar analysis can be performed for a negative wave such that $\phi \leq \min(2r^-_{i+1}, 2)$, where r^-_{i+1} is defined below.

The general high-order TVD scheme for a nonuniform grid which is used here can be written as

$$u_i^{n+1} = u_i^n - \mu(\Delta_- h_{i+\frac{1}{2}} + \Delta_-((\beta^+_{i+\frac{1}{2}}\phi(r_i^+) - \beta^-_{i+\frac{1}{2}}\phi(r^-_{i+1}))\Delta u^n_{i+\frac{1}{2}})), \tag{3.22a}$$

where $h_{i+\frac{1}{2}}$ is the first-order flux (3.8) and

$$\begin{aligned}
\beta^+_{i+\frac{1}{2}} &= (1 - \nu^+_{i+\frac{1}{2}})a^+_{i+\frac{1}{2}}/2,\\
\beta^-_{i+\frac{1}{2}} &= (1 + \nu^-_{i+\frac{1}{2}})a^-_{i+\frac{1}{2}}/2,\\
\nu^\pm_{i+\frac{1}{2}} &= \tfrac{1}{2}\,(\nu_{i+\frac{1}{2}} \pm |\nu_{i+\frac{1}{2}}|),\\
\mu &= \Delta t/\Delta x_i.
\end{aligned} \tag{3.22b}$$

$\nu_{i+\frac{1}{2}}$ is defined in (3.15) together with r_i^+. When the wave has a negative sign,

$$r^-_{i+1} = \frac{(1 + \nu_{i+\frac{3}{2}})a_{i+\frac{3}{2}}\Delta u_{i+\frac{3}{2}}}{(1 + \nu_{i+\frac{1}{2}})a_{i+\frac{1}{2}}\Delta u_{i+\frac{1}{2}}}$$

and $\phi(r)$ can be any of the usual flux limiters ranging from minmod to superbee [4].

4. Accuracy on a nonuniform grid. In Appendix A we show that while the low- and high-order schemes of (3.6)–(3.8) have a local truncation error of $O(1)$ on a nonuniform grid, the error in the solution converges with $O(\Delta x)$ and they are therefore supraconvergent schemes [22].

While this result holds for an arbitrary fixed nonuniform grid, we stress that the grids used in this paper are obtained via an equidistribution principle. The grid continually adapts with the solution such that the finest grid zones overlay the regions of the flow field with steep flow gradients. Away from the interfaces between grids with differing levels of refinement the grid is uniform and, therefore, the local truncation error of the scheme is restored to the uniform grid value, typically $O(\Delta x^2)$.

Therefore, although formally, only first-order convergence can be demonstrated, the adaptivity can be expected to provide a better convergence rate than that for a fixed arbitrary nonuniform grid.

5. Roe scheme for systems of conservation laws. The extension of the scalar algorithms described above to systems of equations is performed by applying a scalar algorithm to each characteristic equation obtained by decomposition. The main difficulty with this approach is ensuring that conservation is obtained upon recomposition to the conservative variables.

In Roe's scheme [1] for solving the system of conservation laws

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial \boldsymbol{f}(\boldsymbol{u})}{\partial x} = 0 \tag{5.1}$$

with initial data

$$\boldsymbol{u}(x,0) = \boldsymbol{u}_0(x),$$

a linearized equation is used to approximate the solution to the Riemann problem. A constant mean-value Jacobian $\tilde{\boldsymbol{A}}(\boldsymbol{u}_L, \boldsymbol{u}_R)$ is constructed such that the following three properties are satisfied:

(i) $\tilde{\boldsymbol{A}}(u_L, u_R)$ is consistent, as $\boldsymbol{u}_L \to \boldsymbol{u}_R$ $\tilde{\boldsymbol{A}}(\boldsymbol{u}_L, \boldsymbol{u}_R) \to \boldsymbol{A}(\boldsymbol{u})$, where $\boldsymbol{A}(\boldsymbol{u}) = \boldsymbol{f}_{\boldsymbol{u}}$;
(ii) For any $\boldsymbol{u}_L$, $\boldsymbol{u}_R$, $\tilde{\boldsymbol{A}}(\boldsymbol{u}_L, \boldsymbol{u}_R)\ (\boldsymbol{u}_R - \boldsymbol{u}_L) = \boldsymbol{f}(\boldsymbol{u}_R) - \boldsymbol{f}(\boldsymbol{u}_L)$;
(iii) The eigenvectors of $\tilde{\boldsymbol{A}}$ are linearly independent.

Condition (ii) ensures that the resulting scheme is conservative; (ii) and (iii) ensure that in the case of a single shock wave, the solution to the Riemann problem will be exact [1]. In order to apply the scalar scheme, the system is decomposed by expanding $\Delta \boldsymbol{u}$ as

$$\Delta \boldsymbol{u} = \sum_j \boldsymbol{e}_j \alpha_j, \tag{5.2}$$

and the flux difference $\Delta \boldsymbol{f}$ as

$$\Delta \boldsymbol{f} = \tilde{\boldsymbol{A}} \Delta \boldsymbol{u} = \sum_j \lambda_j \boldsymbol{e}_j \alpha_j, \tag{5.3}$$

where $\Delta \boldsymbol{u}$ and α_j are the changes in the conservative and characteristic variables, respectively, and λ_j and $\boldsymbol{e}_j$ are the eigenvalues and eigenvectors of $\tilde{\boldsymbol{A}}$, respectively.

Condition (ii) enables the first-order scheme to be written as

$$\boldsymbol{u}_i^{n+1} = \boldsymbol{u}_i^n - \mu(\boldsymbol{h}_{i+\frac{1}{2}} - \boldsymbol{h}_{i-\frac{1}{2}}), \tag{5.4a}$$

where now

$$\boldsymbol{h}_{i+\frac{1}{2}} = \left(\frac{\boldsymbol{f}_i + \boldsymbol{f}_{i+1}}{2}\right) - \frac{1}{2}\sum_j \boldsymbol{e}_j |\lambda_j| \alpha_j. \tag{5.4b}$$

This scheme is immediately applicable to nonuniform grids provided that the grid ratio μ defined in (3.22) is used in (5.4). The high-order version of this scheme can be derived as in the scalar case (3.22), by the addition of an antidiffusive flux to the first-order flux $\boldsymbol{h}_{i+\frac{1}{2}}$, resulting in

$$\boldsymbol{u}_i^{n+1} = \boldsymbol{u}_i^n - \mu\Delta_- \left(\boldsymbol{h}_{i+\frac{1}{2}} + \sum_j \boldsymbol{e}_j \left(\beta^{j^+}\phi\left(r_i^{j^+}\right) - \beta^{j^-}\phi\left(r_{i+1}^{j^-}\right)\right)\alpha_{i+\frac{1}{2}}^j\right), \tag{5.5}$$

where a superfix j represents the jth characteristic component; each component has its own wave speed λ^j and gradient α^j, and these replace the scalar wave speed a and the gradient Δu, respectively, in (3.15) and (3.22).

6. Two-dimensional scheme.

6. Two-dimensional scheme. Consider a system of conservation laws in two dimensions written in integral conservation form

$$\int_\omega (\boldsymbol{u}_t + \boldsymbol{f}_x + \boldsymbol{g}_y) d\tau = 0 \tag{6.1}$$

with initial data

$$\boldsymbol{u}(x,y,0) = \boldsymbol{u}_0(x,y).$$

In order to solve (6.1), the scheme of §5 is extended into two dimensions for an arbitrary, quadrilateral grid. Assuming that the conservation variables $\boldsymbol{u}$ have a piecewise constant variation over each element, an application of the Gauss flux theorem to (6.1) over a given element results in

$$\begin{aligned} \tau_e u_t &= -\sum_k \oint (f dy_k - g dx_k) \\ &= -\sum_k \oint \boldsymbol{F}_k d\ell_k, \end{aligned} \tag{6.2}$$

where

$$\boldsymbol{F} = \boldsymbol{f} c_k + \boldsymbol{g} s_k,$$

$$c_k = dy_k/\ell_k, \qquad s_k = -dx_k/\ell_k,$$

$$\ell_k^2 = dx_k^2 + dy_k^2,$$

(c_k, s_k) are the direction cosines of the kth outward normal, and τ_e is the element (e) area. By further assuming a one-dimensional variation in the variables $\boldsymbol{u}$ across each element face k, the resolved flux $\boldsymbol{F}_k$ can be regarded as being locally one-dimensional. Thus the scheme of §5 can be applied with the Roe flux (corresponding to F) substituted in the above summation (6.2) for each face of the element. The local Jacobian matrix is given by

$$\boldsymbol{A}(\boldsymbol{u}) = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} c + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} s, \tag{6.3a}$$

and the corresponding eigenvectors and eigenvalues of this matrix (which appear in the Roe flux) must satisfy

$$\Delta \boldsymbol{u} = \sum_j \boldsymbol{e}_j \alpha_j, \qquad \Delta \boldsymbol{F} = \sum_j \lambda_j \boldsymbol{e}_j \alpha_j. \tag{6.3b}$$

Finally this scheme can be written in the classical finite-volume form as

$$\tau_e(\boldsymbol{u}_e^{n+1} - \boldsymbol{u}_e^n) = -\Delta t \left(\sum_i (\bar{\boldsymbol{f}}_i \Delta y_i - \bar{\boldsymbol{g}}_i \Delta x_i) - \sum_i \frac{\ell_i}{2} \left(\sum_j \boldsymbol{e}_j |\lambda_j| \alpha_j \right)_i \right), \tag{6.4a}$$

where

$$\bar{\boldsymbol{f}}_i = \tfrac{1}{2}\,(\boldsymbol{f}_e + \boldsymbol{f}_{e_i}), \qquad \bar{\boldsymbol{g}}_i = \tfrac{1}{2}\,(\boldsymbol{g}_e + \boldsymbol{g}_{e_i}), \tag{6.4b}$$

and the summation index i varies over the sides of the element e, $(\Delta x_i, \Delta y_i)$ being the vector element of the length tangential to side i and e_i being the ith neighboring element number.

The high-order version of this scheme can also be written in the form (6.4a). The antidiffusive flux of (5.5) is generalized into two dimensions by associating an antidiffusive flux contribution with each face of the element. The local eigenvalues and eigenvectors of (6.3) are used in the construction of the antidiffusive flux at each face. The flux limiters corresponding to the even- and odd-numbered faces of the element (Fig. 2) are evaluated with respect to the underlying x and y directions of the grid, respectively. Each antidiffusive term is then multiplied by its corresponding element face length and the sign of the local outward normal vector before being assembled on the right-hand side of (6.4).

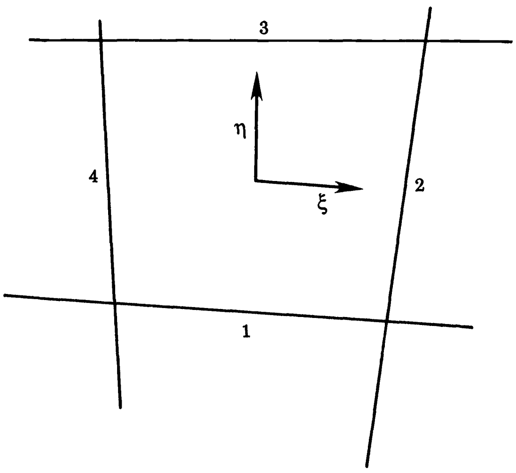


FIG. 2. *Local coordinate system.*

For a regular grid, the resulting scheme is second-order accurate with respect to the $x - y$ coordinate directions. Formal second-order accuracy, including cross-derivative terms, can be achieved via the time-splitting technique of Strang [23].

The Roe matrix A, together with the eigenvectors, eigenvalues, and characteristic increments α_j, are presented in Appendix B for the Euler equations of compressible flow in two dimensions.

7. Adaptive refinement and unrefinement. In this section we shall briefly describe some of the principle features pertaining to h-refinements which are contained within the general data structure developed by Demkowicz and Oden [2] for quadrilateral grids. A description of the logic and account of the structure is given by Demkowicz in [24].

Following the nomenclature of Demkowicz and Oden [2], when a refinement of a quadrilateral element takes place the element is divided into four subelements, which are called "sons." The group of four sons is called a "family" and the original element is called the "father" element (Fig. 3).

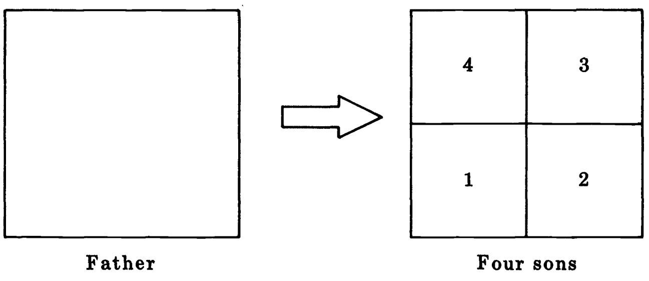


FIG. 3. *Initial refinement.*

A typical refinement, shown in Fig. 4, gives rise to nodes such as p which are called hanging nodes or irregular nodes. Whereas each regular node is located at the corner in common with all of its neighboring elements, the irregular node is not located at a corner for all of its neighboring elements. In this data structure only one irregular node per element side is permitted.

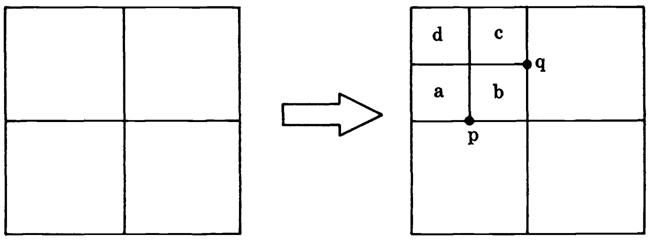


FIG. 4. *Refinement with irregular nodes p, q.*

Due to this constraint (one irregular node per side) any element can have a maximum of eight sides and eight neighbors. Following Devloo [25] the sides of the element are labeled one to eight in the sequence shown in Fig. 5.

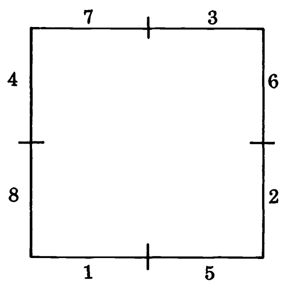


FIG. 5. *Element side numbering.*

This numbering sequence gives rise to some attractive logical relations which are exploited within the data structure [24], [25] and within the extension of the above two-dimensional scheme to an unstructured grid.

Returning to Fig. 4, if one of the sons (b) of the family is refined, the constraint (one irregular node per element side) causes the neighboring element to be refined so that some spreading of the refinement process can occur. This is illustrated in Fig. 6.

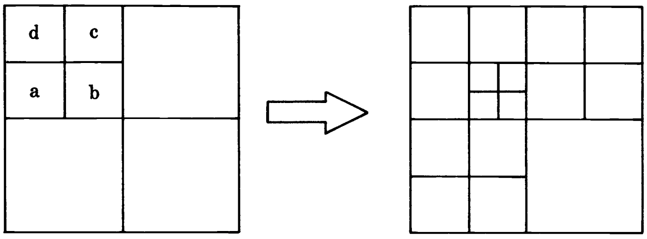


FIG. 6. *Spreading of refinement.*

Adaptive strategy. As in [9], a crude computational grid is generated by dividing the flow domain into a suitably small number of quadrilaterals. The initial crude grid is then globally refined to create a reasonable coarse grid (e.g., Fig. 12), which is used for the first part of the computation. After a specified number of timesteps (or iterations) the adaptive strategy (described below) is invoked.

The adaptive criteria for deciding when to refine an element and unrefine a family is based on monitoring the density gradient as in [9] and [10]. The density is chosen as the key variable because it is discontinuous at shocks and contact discontinuities.

The actual adaptive strategy is listed below. The condition on the gradient of the sons of the family in (iii)(a) makes the procedure much more robust.

(i) Define the refinement and unrefinement tolerances (user specified constants) $\in_r$, $\in_u$, respectively.

(ii) Find the maximum L_∞ density gradient throughout the field,

$$\|\nabla\rho\|_\infty = \max_{1\le NEL \le NRELEM} |\nabla\rho|_{NEL}.$$

(iii) Looping over the families (N_F = I,NRFAM), find the density gradient modulus $|\nabla\rho|_{e_i}$ of each son $e_i (i = 1, 4)$ of the family (number N_F). Define the mean family density gradient by

$$|\nabla\rho|_{N_F} = \tfrac{1}{4}\sum_{i=1}^{4} |\nabla\rho|_{e_i}.$$

If for each son (e_i) of the family (N_F),

$$\text{(a)}\quad |\nabla\rho|_{e_i} < \in_r \|\nabla\rho\|_\infty \text{ and}$$

$$\text{(b)}\quad |\nabla\rho|_{N_F} < \in_u \|\nabla\rho\|_\infty,$$

then unrefine the family and define the new element vector of conservation variables by

(7.1) $$u_{N_F} = \sum_{i=1}^{4} A_{e_i} u_{e_i} \Big/ \sum_{i=1}^{4} A_{e_i},$$

where u_{e_i} are the conservative variables of the ith son of the family N_F and A_{e_i} is the area of the ith son (element).

(iv) Looping over the elements (NEL = 1,NRELEM), find the density gradient modulus $|\nabla\rho|_{NEL}$ of each element NEL. If

$$|\nabla\rho|_{NEL} > \epsilon_r \, \|\nabla\rho\|_\infty,$$

refine the element NEL to produce four sons $e_i (i = 1, 4)$. The conservation variables for each of the four sons is defined by the value of the father, i.e.,

(7.2) $$u_{e_i} = u_{N_{EL}}.$$

The use of (7.1) and (7.2) ensure that conservation is maintained after a sequence of unrefinements and refinements.

The use of (7.2) is the most diffuse option. A bilinear interpolation would be more accurate, but then conservation would be more difficult to enforce.

8. Extension of the Roe scheme to an unstructured grid. The low-order scheme described in §6 is relatively easy to extend to an unstructured grid of the type discussed above. For an element-wise implementation of the scheme, instead of looping over sides one to four, as in the case of a structured grid, it is convenient to loop over sides one to eight, and test for the existence of a neighbor for any side N, say, between five and eight ($5 \le N \le 8$). If there is no neighbor, go to the end of the loop, otherwise the lengths Δx, Δy, ℓ defined in §6 are halved before performing the flux calculations of §6 for sides N and $N - 4$.

For side $N - 4$ the flux calculation will involve quantities at i and i_2 while for side N the flux calculation will involve quantities at i and i_6 (see Fig. 7).

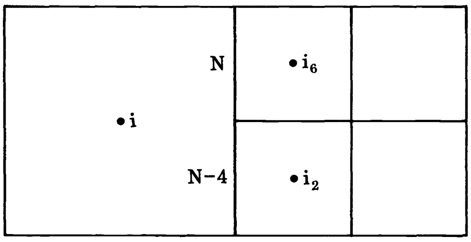


FIG. 7. *Flux calculation on unstructured grids.*

The extension of the higher-order scheme is far more complicated. The support of the scheme in the case of a structured grid is sketched in Fig. 8. Unlike the low-order scheme, which involves five elements, the higher-order scheme involves nine elements, the additional four elements being "neighbors of the neighbors" of the element in question.

The higher-order upwinding can be conveniently obtained on a structured grid where global curvilinear ξ and η "directions" may be identified and the flux limiting performed along these "directions."

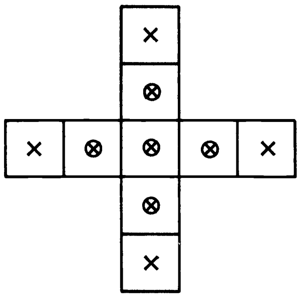


FIG. 8. *Support of two-dimensional first-order scheme* o *and second-order scheme* x.

Turning now to the case of an "unstructured grid" of the type discussed above, it is easy to see that while the grid is unstructured in the sense that certain grid lines terminate in the field, the grid still retains a certain structure in the sense that global ξ and η "directions" can still be identified throughout the grid. This property is completely exploited by constructing directional flux limiters corresponding to the even-numbered sides (ξ direction) and odd-numbered sides (η direction) of the element, respectively. (This extension of the flux-limited scheme would not be applicable to an unstructured triangular grid such as that generated in [10].)

For illustration we shall consider the construction of the flux limiters in the ξ direction (corresponding to a positive wave) which contribute to the solution at the central element (i). Various grid configurations that may occur in the "ξ direction" which affect the positive wave limiters are sketched in Fig. 9.

We introduce a notation (consistent with Fig. 6) that relates the standard flux limiters (§3) in the ξ direction to the even-numbered sides of the element, and that denotes the flux limiters corresponding to sides 2, 4, 6, and 8 by, ϕ_2, ϕ_4, ϕ_6, and ϕ_8, respectively. For a regular (structured) grid (Fig. 9(a)), the solution at element i will involve the usual flux limiters $\phi_2 = \phi(r_i^+)$ and $\phi_4 = \phi(r_{i-1}^+)$, where r_i^+ is defined in §4 as

$$r_i^+ = \frac{\nu_{i-\frac{1}{2}}(1-\nu_{i-\frac{1}{2}})(u_i - u_{i-1})}{\nu_{i+\frac{1}{2}}(1-\nu_{i+\frac{1}{2}})(u_{i+1} - u_i)}. \tag{8.1}$$

We shall adopt the more general notation

$$r(u_{i-1}, u_i, u_{i+1}) = r_i^+. \tag{8.2}$$

In the second configuration (Fig. 9(b)), ϕ_2, ϕ_4, and ϕ_6 now make contributions to the solution at element i, where now

$$\begin{aligned} \phi_2 &= \phi(r(u_{i-1}, u_i, u_{i_2})), \\ \phi_4 &= \phi(r(u_{i-2}, u_{i-1}, u_i)), \\ \phi_6 &= \phi(r(u_{i-1}, u_i, u_{i_6})), \end{aligned} \tag{8.3}$$

where i_2 and i_6 are the right-hand element numbers (Fig. 9(b)).

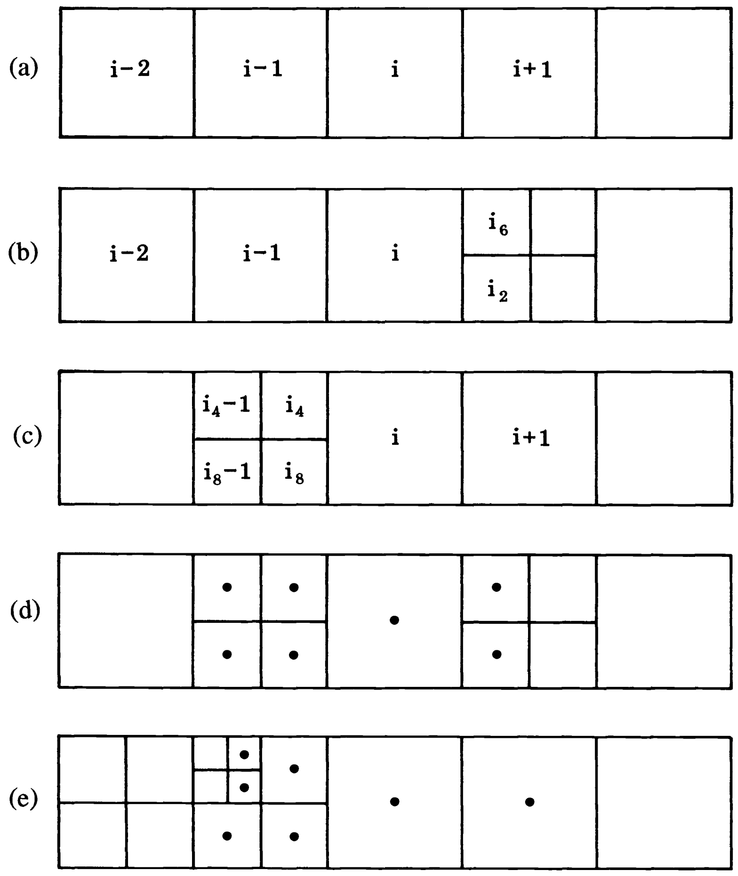


FIG. 9. *Adaptive support of scheme (ξ direction; $\nu > 0$).*

In the third configuration (Fig. 9(c)), ϕ_2, ϕ_4, and ϕ_8 make contributions to element i. In this case ϕ_2 is a function of both $r(u_{i_4}, u_i, u_{i+1})$ and $r(u_{i_8}, u_i, u_{i+1})$. As a general rule, whenever the flux limiter is a function of two ratios (as in this case), we shall define the limiter to be a minimum of the two possibilities. Therefore,

$$\begin{aligned} &\phi_2 = \min\left(\phi(r(u_{i_4}, u_i, u_{i+1})),\ \phi(r(u_{i_8}, u_i, u_{i+1}))\right), \\ (8.4) \qquad &\phi_4 = \phi(r(u_{i_{4-1}}, u_{i_4}, u_i)), \\ &\phi_8 = \phi(r(u_{i_{8-1}}, u_{i_8}, u_i)). \end{aligned}$$

In the fourth configuration (Fig. 9(d)), all of ϕ_2, ϕ_4, ϕ_6, and ϕ_8 make a contribution to element i. As in the third case, ϕ_2 is chosen as

$$\phi_2 = \min\left(\phi(r(u_{i_4}, u_i, u_{i_2})),\ \phi(r(u_{i_8}, u_i, u_{i_2}))\right).$$

Similarly, since ϕ_6 is now a function of two ratios, then

$$\phi_6 = \min\left(\phi(r(u_{i_4}, u_i, u_{i_6})),\ \phi(r(u_{i_8}, u_i, u_{i_6}))\right),$$

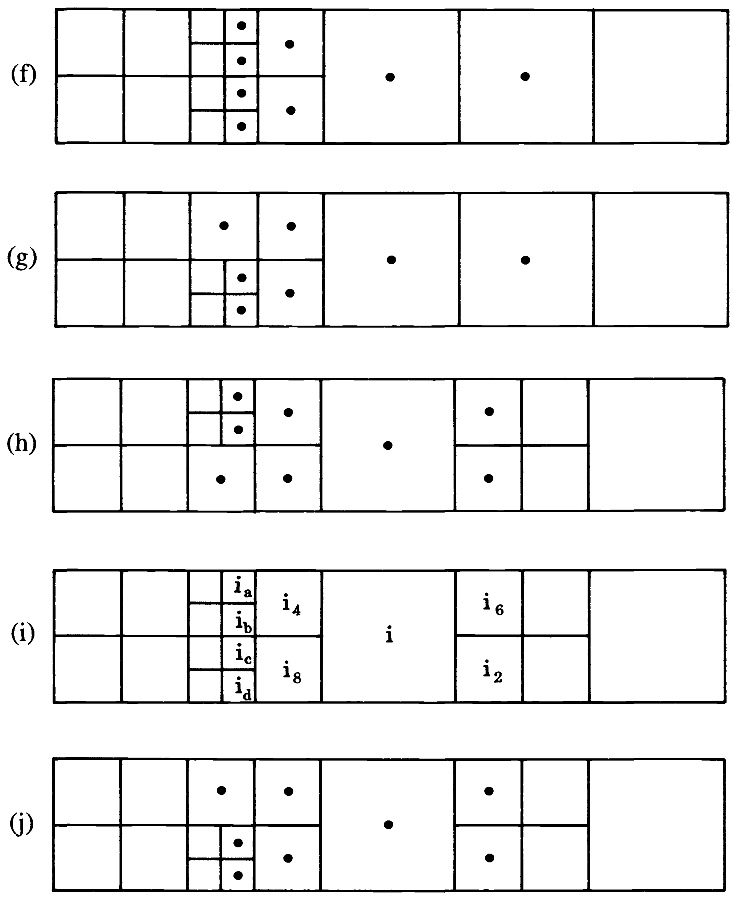


FIG. 9 (*continued*).

while the limiters ϕ_4 and ϕ_8 are evaluated as in the third case.

In the configurations in Figs. 9(e)–(g) the evaluation of ϕ_2 is identical to the third case described above, while for the configurations in Figs. 9(h)–(j) the evaluations of ϕ_2 and ϕ_6 are identical to the fourth case described above.

In configurations (e)–(j), ϕ_4 and/or ϕ_8 are functions of two ratios and we continue to apply the above rule, for example, in Fig. 9(i),

$$\phi_4 = \min(\phi(r(u_{i_a}, u_{i_4}, u_i)),\ \phi(r(u_{i_b}, u_{i_4}, u_i))),$$
$$\phi_8 = \min(\phi(r(u_{i_c}, u_{i_8}, u_i)),\ \phi(r(u_{i_d}, u_{i_8}, u_i))).$$

A similar analysis is performed for a negative wave with each configuration (e)–(j) inverted with extra refinements now appearing in the right-hand element of the support. Finally, the same analysis is performed for both positive and negative waves in the η direction, where the limiters on the upper side of the element are ϕ_3 and ϕ_7, while the limiters on the lower side of the element are ϕ_1 and ϕ_5.

9. Grid movement. The grid movement considered in this work is strictly for steady-state calculations and is induced via an equidistribution scheme constructed using

$$r_i^{(n+1)} = \sum_e \frac{r_e^n |\nabla\rho|_e}{\sum_e |\nabla\rho|_e}, \tag{9.1}$$

where $\underline{r}_e$ are the position vectors of the centers of gravity of the elements surrounding node i (with position vector $\underline{r}_i$), and the summation is performed over these elements. Similar approaches have been used in [9] and [26].

As noted in [26], the scheme (9.1) can be identified as an equidistribution technique in the limit as the iterative cycle converges. Consider the x component of (9.1) written as

$$x_i^{n+1} - x_i^n = \frac{\sum x_e^n w_e}{\sum w_e} - x_i^n, \tag{9.2}$$

where $w_e = |\nabla\rho|_e$. In one dimension the centers of the elements which surround node i are

$$x_{e_1} = \tfrac{1}{2}(x_i + x_{i-1}), \qquad x_{e_2} = \tfrac{1}{2}(x_{i+1} + x_i), \tag{9.3}$$

and

$$w_{e_1} = w_{i-\frac{1}{2}}, \qquad w_{e_2} = w_{i+\frac{1}{2}}. \tag{9.4}$$

Equations (9.2), (9.3), and (9.4) give

$$x_i^{n+1} - x_i^n = \frac{(x_{i+1}^n - x_i^n)w_{i+\frac{1}{2}} - (x_i^n - x_{i-1}^n)w_{i-\frac{1}{2}}}{(w_{i+\frac{1}{2}} + w_{i-\frac{1}{2}})}, \tag{9.5}$$

and as $x_i^{n+1} - x_i^n \to 0$, we obtain

$$\Delta x_{i+\frac{1}{2}} w_{i+\frac{1}{2}} = \Delta x_{i-\frac{1}{2}} w_{i-\frac{1}{2}} = \text{ constant.} \tag{9.6}$$

After some experimentation it was found that a reasonably robust equidistribution scheme can be defined as

$$r^{n+1} = \frac{\sum_e r_e^n((|\nabla\rho|_e/\|\nabla\rho\|_\infty)A_e\alpha) + \beta}{\sum_e((|\nabla\rho|_e/\|\nabla\rho\|_\infty)A_e\alpha) + \beta}, \tag{9.7}$$

where α is a user-defined constant which determines the strength of the grid movement, and $\beta(= 10^{-4})$ is a regularization constant which prevents any singularity from occurring (in (9.7)). A_e is the Jacobian of the eth element.

10. Results. The adaptive schemes described above are used to solve the Euler equations of compressible flow, written in conservation form

$$\boldsymbol{u}_t + \boldsymbol{f}_x + \boldsymbol{g}_y = \boldsymbol{0}, \tag{10.1}$$

where

$$\begin{aligned} \boldsymbol{u} &= (\rho,\ \rho u,\ \rho v,\ E)^T, \\ \boldsymbol{f} &= \Big(\rho u,\ \rho u^2 + p,\ \rho uv,\ u(E+p)\Big)^T, \\ \boldsymbol{g} &= \Big(\rho v,\ \rho uv,\ \rho v^2 + p, v(E+p)\Big)^T. \end{aligned} \tag{10.2}$$

Here ρ, p, and E are the density, pressure, and energy per unit volume for an ideal gas, (u, v) are the cartesian components of velocity, and

$$E = \rho\left[\tfrac{1}{2}(q^2) + p/\rho(\gamma - 1)\right],$$

γ being the ratio of specific heat capacities and $q^2 = u^2 + v^2$.

Three kinds of physical boundary conditions are imposed, namely, (i) supersonic inflow, (ii) supersonic outflow, and (iii) solid wall.

Since the scheme is cell-centered, a band of dummy nodes is required at all boundaries to complete the definition of the discretization of the scheme.

In condition (i) all conservation variables are specified at the dummy nodes with their free-stream values

$$u_{\text{dummy}} = u_{\text{free-stream}}.$$

In condition (ii) the dummy values are found by assuming that the normal gradients of the flow variables are zero at the outflow boundary, so that $\partial \underline{u}/\partial n = 0$, which results in

$$u_{\text{dummy}} = u_{\text{interior}}.$$

In condition (iii) at a solid wall, the normal velocity is set equal to zero while reflection conditions are used for the density ρ, energy E, and tangential velocity v^t with

$$\frac{\partial \rho}{\partial n} = \frac{\partial E}{\partial n} = \frac{\partial v^t}{\partial n} = 0,$$

where the tangential velocity v^t is defined by $v^t = \underline{q} \cdot \hat{\underline{t}}$ and q is the velocity vector. A second layer of dummy values are introduced (via reflection) at a solid wall to complete the support of the second-order scheme.

We present the results obtained for three test cases. In all cases the computed density contours will be shown for 30 uniform intervals both for uniform and adaptive grid computations, respectively.

For the first two cases involving steady-state flow, three kinds of adaptive strategy are tested: (i) grid movement using equidistribution; (ii) grid refinement/unrefinement; and (iii) grid refinement and movement.

10.1. Supersonic flow over a wedge. The wedge has an inclination of 20° to the horizontal. Initially the flow is assumed to be uniform throughout the field with a free-stream Mach number $M_\infty = 3.0$. An exact solution for this problem can be obtained [27], which consists of a uniform shock wave extending from the compression corner into the flow field at an angle of 37.5°.

The result obtained using a uniform 32 × 16 grid Fig. 10(a) is shown in Fig. 10(b). The exact solution lies within the band of density contours.

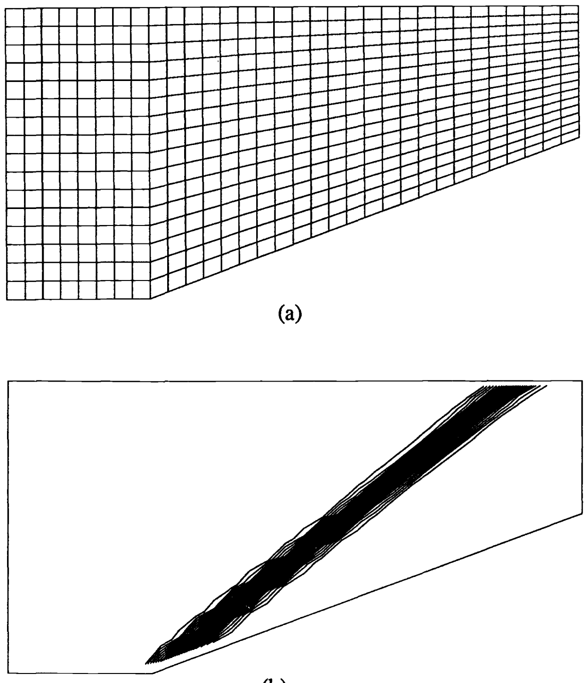


FIG. 10. *Wedge—regular grid.*

Using the uniform grid solution as an initial data for the equidistribution scheme (9.7), the solution and grid shown in Fig. 11 are obtained by applying (9.7) for three iterations, each iteration being performed after 300 steps, with $\alpha = 4.3$. A considerable improvement in shock resolution is obtained which may be due not only to the local reduction in the size of the element in the shock region, but also to the improvement in the local orientation of the grid relative to the shock.

Next we consider the h-refinement strategy of §7. An initial solution is obtained on a 16×8 grid (Fig. 12). The solution (and grid) obtained after two applications of the h-refinement strategy of §7 is shown in Fig. 13, and after a third application, in Fig. 14. Each application was made after 200 steps. The refinement and unrefinement tolerances are $\varepsilon_r = 0.16$ and $\varepsilon_u = 0.16$, respectively. The final solution is well aligned with the exact solution, and very good resolution of the shock has been obtained.

Finally, the combination refinement and movement $(h - r)$ is tested using the solution of Fig. 13 as an initial data; the result shown in Fig. 15 is obtained after three movements, one every 300 steps, with $\alpha = 5$.

10.2. Blunt body. The second steady-state problem that we consider is a blunt body placed in a supersonic flow field with free-stream Mach number $M_\infty = 6.57$ and $\gamma = 1.38$ at 0° angle of attack. The larger (extended body) version of this problem has been studied by Oden, Strouboulis, and Devloo [9] and Bey et al. [28].

An initial skeleton grid of 5×5 elements is generated and globally refined twice to produce a 20×20 uniform grid, which is smoothed by using (9.7) with the density gradient removed.

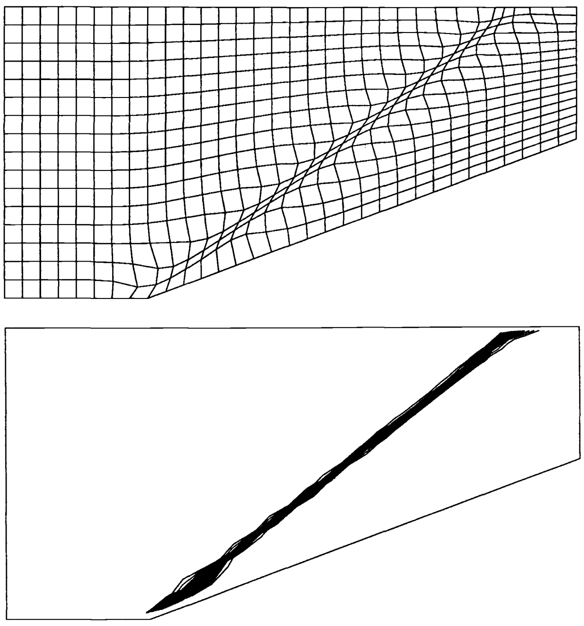


FIG. 11. *Wedge—regular grid with node relocation.*

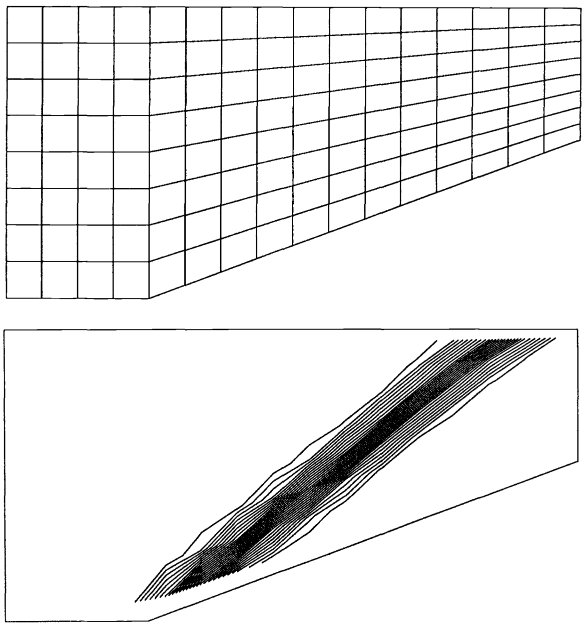


FIG. 12. *Wedge—initial coarse grid.*

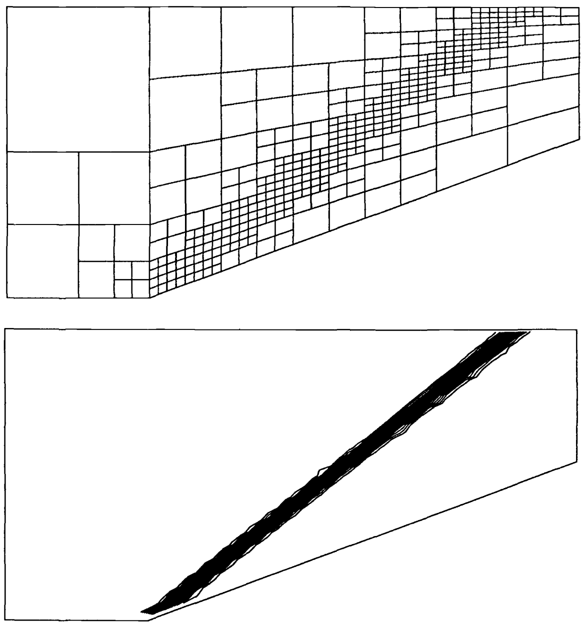


FIG. 13. *Wedge—with five grid levels of h-refinement.*

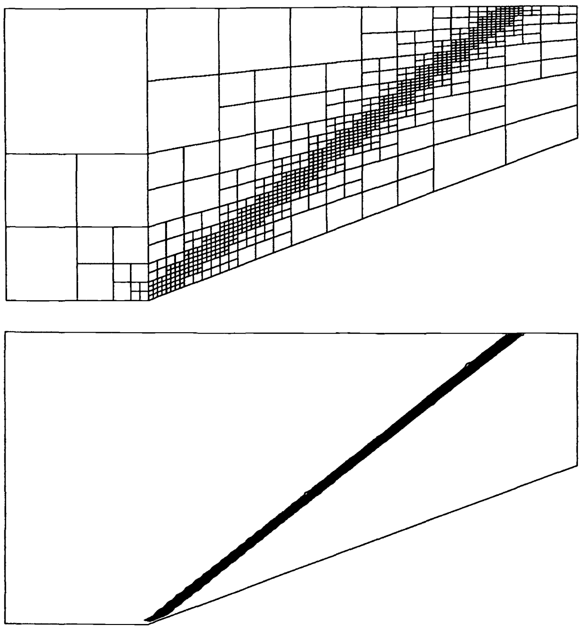


FIG. 14. *Wedge—with six grid levels of refinement.*

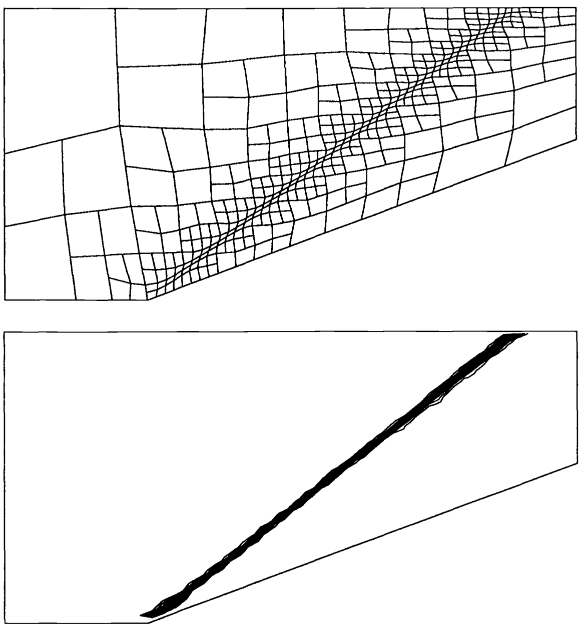


FIG. 15. *Wedge—h-r method.*

The solution obtained on the uniform 20×20 smoothed grid is shown in Fig. 16. The exact shock location of this problem is found following Billig [29] and lies at the center of the computed shock location.

As in the previous case, the uniform grid solution (Fig. 16) is used as an initial data for the equidistribution scheme (9.7). After using (9.7) for two iterations, one every 200 steps, the result in Fig. 17 is obtained. The effect of grid distortion upon the solution error is not analyzed here. However, in both of the steady-flow problems, (9.7) proves to be an extremely effective simple tool for enhancing the initial uniform grid solutions.

Next, the h-refinement strategy of §7 is applied. The refinement algorithm is applied (with $\varepsilon_r = 0.007$, $\varepsilon_u = 0.44$) using the solution in Fig. 16 as an initial data, then applied again after 100 steps and 500 steps. The solution obtained at convergence is shown in Fig. 18 together with the final grid. The shock resolution is much improved and is aligned with the exact solution.

Finally, the combination of h-refinement and movement is tested with six iterations of grid movement, one every 200 steps, with $\alpha = 5.0$. The resulting solution and grid are shown in Fig. 19. The residual $\|\rho^{n+1} - \rho^n\|_\infty$ is reduced to 0 (10^{-5}) by 2000 timesteps. All of the above results were computed with the non-time-split scheme using the van Leer flux limiter [16], [4] with

$$\phi(r) = (r + |r|)/(1 + |r|). \tag{10.3}$$

The superbee limiter [4] was also tried but gave some convergence problems for these cases.

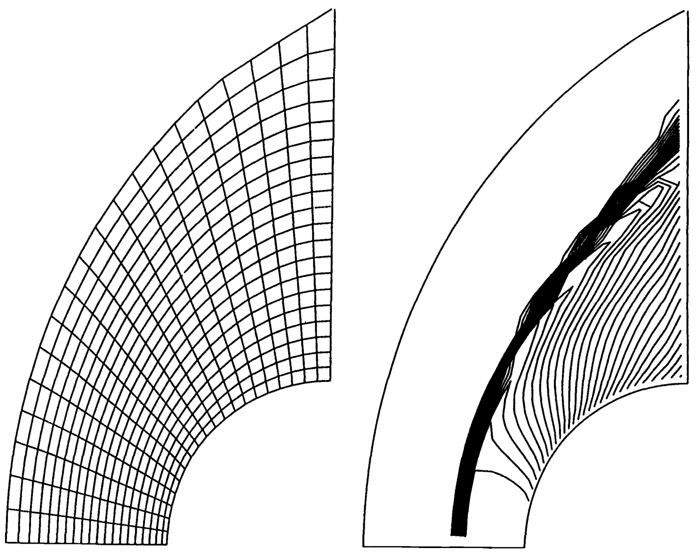


FIG. 16. *Blunt body, regular grid.*

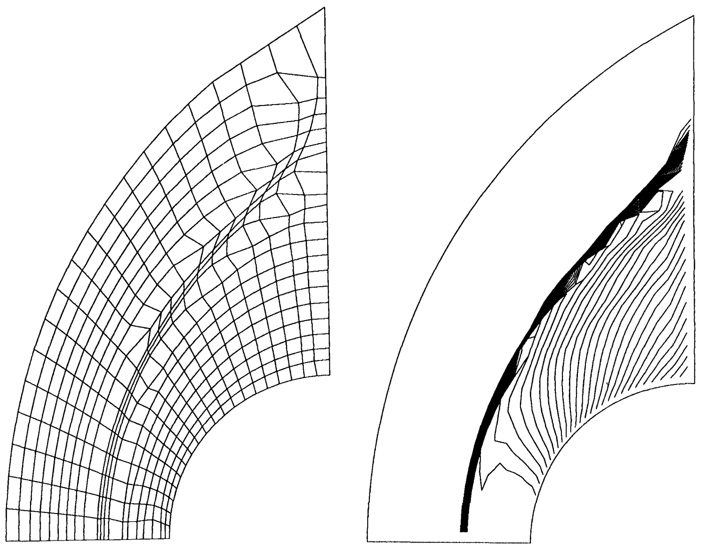


FIG. 17. *Blunt body, regular grid after node relocation.*

10.3. A Mach-3 wind tunnel with a step. This problem, used by Woodward and Colella [8] for comparing a variety of methods, is used here for a test of the transient behavior of the method and solution. The problem begins with a uniform Mach-3 flow throughout the field impinging on a step [8]. The sequence of results

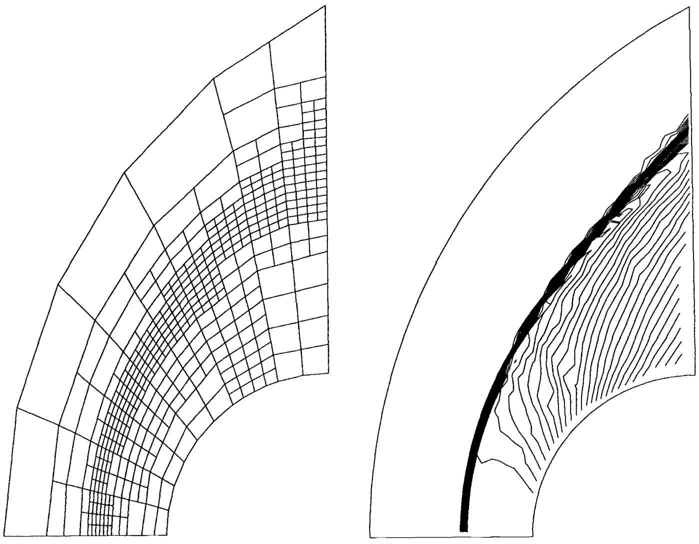


FIG. 18. *Blunt body, h-refinement.*

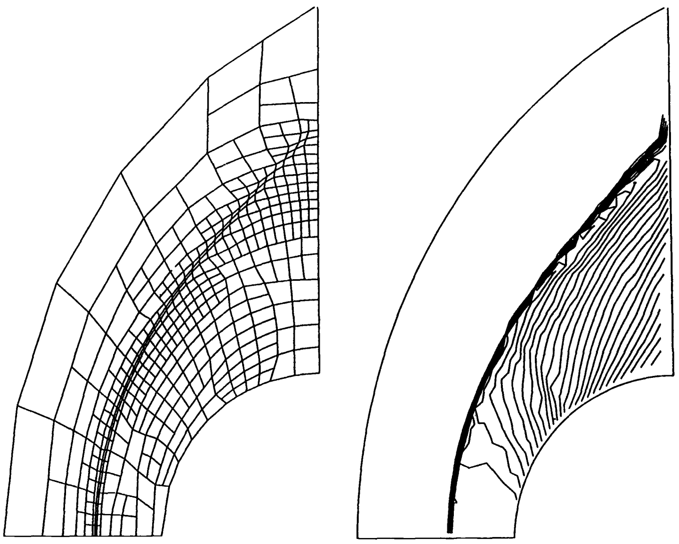


FIG. 19. *Blunt body, h-r method.*

obtained using the time-split scheme, together with the resulting adaptive grids, is shown in Figs. 20–24 for output times $t = 0.5, 1, 2, 3, 4$. These results were obtained by starting on a uniform $(\Delta x = \Delta y = 1/20)$ grid and applying the refinement algorithm in the sequence given in §7 after every 25 timesteps.

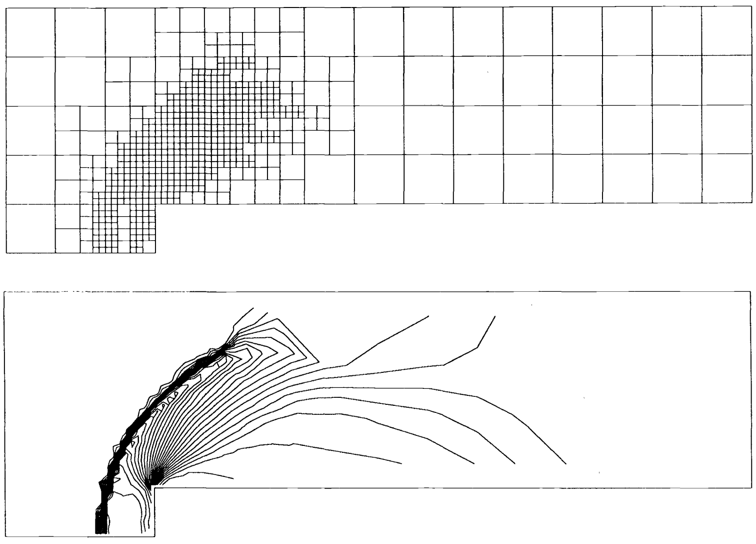


FIG. 20. $t = 0.5$, *fully second-order time-split scheme.*

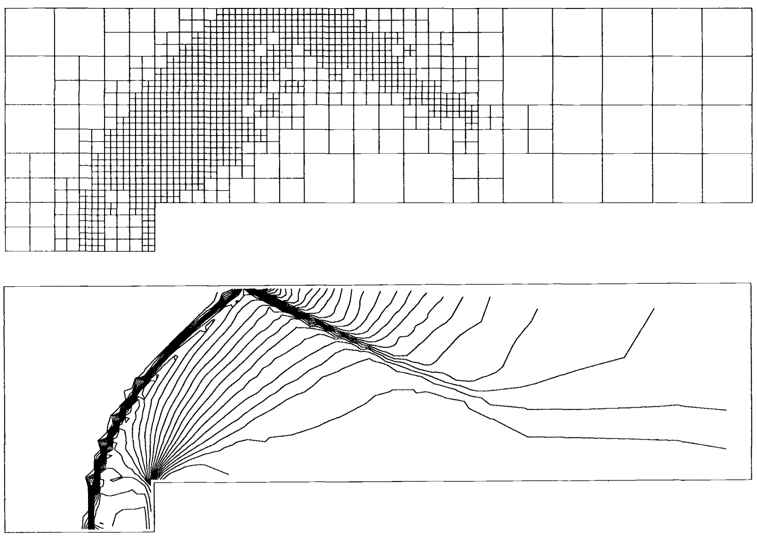


FIG. 21. $t = 1.0$, *fully second-order time-split scheme.*

The condition (iii)(a) of §7 was particularly important in ensuring consistently good results. The tolerances used are $\varepsilon_r = 0.061$ and $\varepsilon_u = 0.44$. The superbee flux limiter was used in all three formulations.

Excellent agreement is obtained between the adaptive- and fixed-grid solutions, e.g., at time $t = 2$ compare Fig. 22 with Fig. 25(a). In the final stage of the computation from $t = 3$ to $t = 4$ the fixed-grid method is unable to sustain the shock reflection

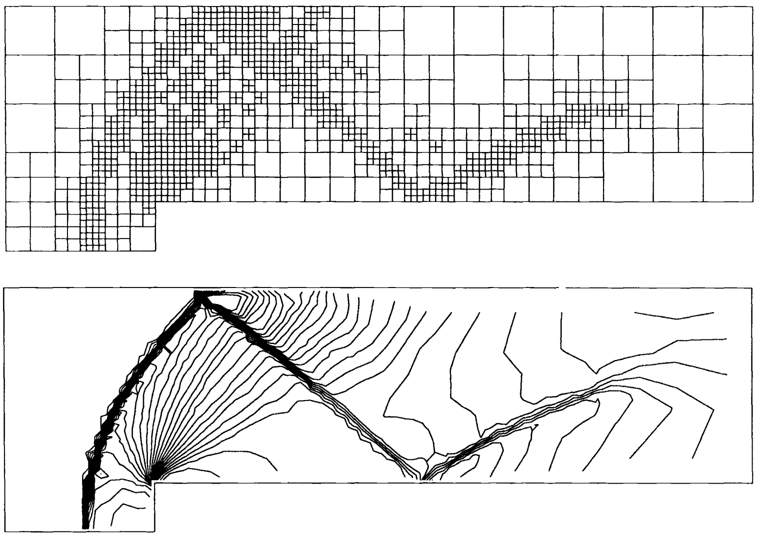


FIG. 22. $t = 2.0$, *fully second-order time-split scheme.*

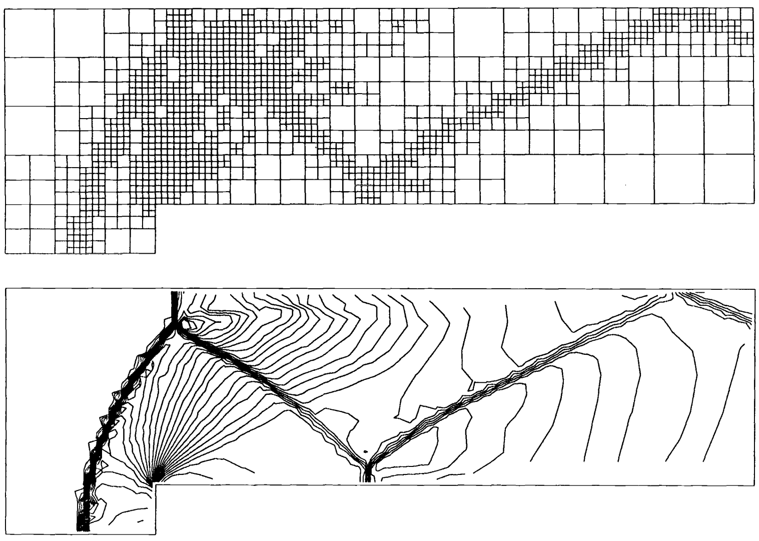


FIG. 23. $t = 3.0$, *fully second-order time-split scheme.*

on the upper wall of the tunnel (Fig. 25(b)), in striking contrast to the adaptive grid result of Fig. 24.

At transition cells (e.g., Fig. 7) the approximation (6.4a) induces truncation errors of $\pm O(1)$ in the two neighboring cells i_2 and i_6, respectively. However, since discontinuities never cross transition zones (due to adaptivity) this potential source of error is avoided. Also this error cancels in the global sum with respect to conservation.

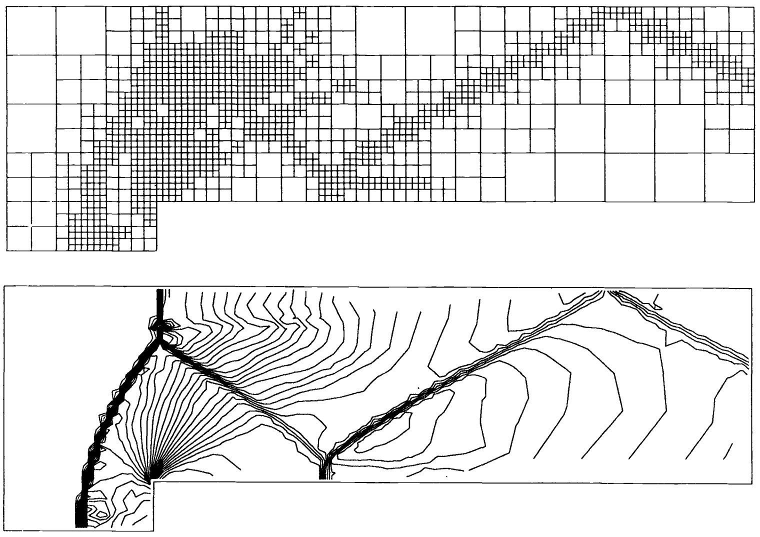
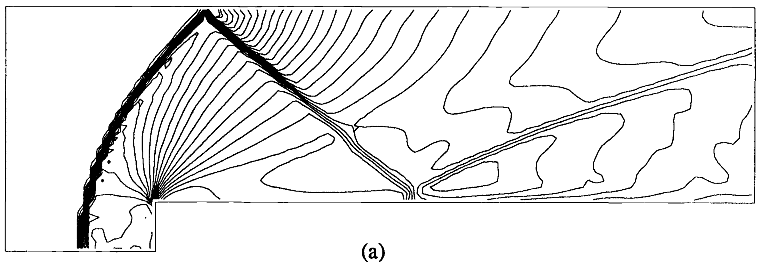
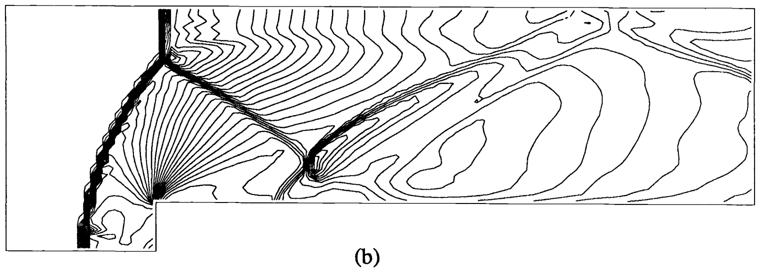


FIG. 24. $t = 4.0$, *fully second-order time-split scheme.*



(a)



(b)

FIG. 25. *Uniform grid computation (time split).* (a) *Time* $t = 2.0$; (b) *Time* $t = 4.0$.

The current version of the computer code has not been optimized in any way and has been written mainly from the point of view of convenience. However, large savings in computer time are obtained in all cases. A typical comparison between an h-adaptive computation and a uniform fine grid (with the same level of refinement, using the same computer code) shows the adaptive method to be about four times faster. This factor tends to increase with grid level, thus, the gains obtained depend considerably upon the adaptive strategies employed, particularly for the steady-state cases.

Timestepping. A uniform timestep (fixed Δt throughout the grid) has been used in all of the work presented here. A point for further investigation would be to combine this method with the adaptive timestepping procedure of Berger [30], where a variable-size timestep is used according to the local level of grid refinement.

There is a striking contrast between the adaptive method of Berger and Oliger [11] and that employed here. Their method for refining the grid involves overlaying sequences of finer grids on the coarse-grid areas which contain steep flow gradients. This method inevitably involves using more elements than are required. Conversely, in the adaptive method used here, only the actual elements which detect large flow gradients are flagged for refinement, although in practice some transition elements are created (see §7). While the former approach is logically more simple than the latter, it would appear that it has a greater need for adaptive timestepping.

In Table 1, we give the grid level and number of elements for the adaptive grids shown in Figs. 14, 18, and 24. An inspection of Table 1, together with the corresponding grids and results, demonstrates that near optimal use has been made of the grid refinements and unrefinements, with the majority of elements packed into the regions of high flow gradients in each case.

TABLE 1
Grid levels and element counts.

	Grid level	Number of elements	Total
Wedge	4×2	1	
Case 1	8×4	10	
	16×8	35	
	32×16	71	
	64×32	161	
	128×64	588	866
Blunt body	5×5	4	
Case 2	10×10	37	
	20×20	119	
	40×40	276	436
Step	15×5	14	
Case 3	30×10	66	
	60×20	265	
	120×40	836	1181

From the table it is possible to deduce the effective advantage of adaptive timestepping. Assuming that the timestep is halved for each fine-grid level created, then for every eight minimum timesteps, Case 3 of Table 1 reveals a potential saving of 16 percent of the run time of the adaptive code. This assumes no extra overhead in performing the adaptive timestepping procedure. For the nine-point schemes used here the support can be across four interfaces simultaneously in one direction (§8), which complicates the logic of adaptive timestepping in two dimensions.

For steady-state problems, nonconservative local timestepping could be used.

However, Table 1 suggests that only a comparatively small reduction in computer time would be obtained in the steady-state cases 1 and 2.

11. Conclusions. A two-dimensional version of the higher-order approximate Riemann solver of Roe [14] is presented in finite-volume form for application to unstructured quadrilateral grids.

A one-dimensional analysis of the solution error indicates that convergence is formally $O(\Delta x)$ for an arbitrary nonuniform grid.

However, the quality of the results and the very fine resolution obtained by the high-order scheme on an adaptive grid is comparable with that obtained by the same scheme on a fixed uniform grid, with a cell size corresponding to that of the finest adaptive grid level.

Great savings in computer time are obtained; the adaptive code is a factor of 4 times faster than the same code run on the corresponding fixed uniform grid.

Appendix A. Question of accuracy on a nonuniform grid. In this appendix we will show that for a sufficiently smooth solution, when the cell-centered low-order upwind scheme is applied to the linear advection equation on a nonuniform grid, the error in the solution is of $O(h)$ while the local truncation error is $O(1)$. When the error in the solution is better behaved than the local truncation error the scheme is supraconvergent [22].

The low-order upwind scheme for linear advection (3.6), (3.8) can be written as

$$u_i^{n+1} = u_i^n - \frac{a\Delta t}{2\Delta x_i}(u_{i+1}^n - u_{i-1}^n) + \frac{|a|\Delta t}{2\Delta x_i}(u_{i+1}^n - 2u_i^n + u_{i-1}^n). \tag{A.1}$$

Substituting the exact solution $u(x,t)$ into (A.1) and performing Taylor series expansions about $u(x_i,t^n)$, the leading truncation error is found to be

$$\tau = au_{x_i}\frac{(\Delta x_{i+\frac{1}{2}} + \Delta x_{i-\frac{1}{2}} - 2\Delta x_i)}{2\Delta x_i} - |a|u_{x_i}\frac{(\Delta x_{i+\frac{1}{2}} - \Delta x_{i-\frac{1}{2}})}{2\Delta x_i} + O(\Delta t, \Delta x), \tag{A.2}$$

where

$$\Delta x_{i+\frac{1}{2}} = \tfrac{1}{2}\,(\Delta x_i + \Delta x_{i+1})$$

and Δx_i is the ith element length.

The local truncation (A.2) is $O(1)$, which suggests that the numerical solution will not converge to the physically correct solution. However, by performing an analysis similar to Kreiss et al. [22] we can show that the error in the solution is of $O(\Delta x)$.

First, we obtain the discrete error equation by subtracting the grid difference equation (A.1) from the truncation error which results in

$$\frac{e_i^{n+1} - e_i^n}{\Delta t} + \frac{a(e_{i+1}^n - e_{i-1}^n)}{2\Delta x_i} - \frac{|a|}{2\Delta x_i}(e_{i+1}^n - 2e_i^n + e_{i-1}^n) = \tau_i. \tag{A.3}$$

Multiplying (A.3) by Δx_i and summing over $i = 1$ to j gives

$$\begin{aligned}\sum_{i=1}^{j}\frac{e_i^{n+1} - e_i^n}{\Delta t}\Delta x_i + \frac{a}{2}(e_j^n + e_{j+1}^n) - \frac{|a|}{2}(e_{j+1}^n - e_j^n)\\ - \frac{a}{2}(e_1^n + e_0^n) + \frac{|a|}{2}(e_1^n - e_0^n) = \sum_{i=1}^{j}\tau_i\Delta x_i;\end{aligned} \tag{A.4}$$

using (A.2) gives

$$\sum_{i=1}^{j} \tau_i \Delta x_i = \tfrac{1}{2} \sum_{i=1}^{j} (a\Delta_-(\Delta x_{i+1} - \Delta x_i) - |a|(\Delta x_{i+\frac{1}{2}} - \Delta x_{i-\frac{1}{2}}))u_{x_i}.$$

Performing a summation by parts results in

$$\text{(A.5)} \qquad \sum_{i=1}^{j} \tau_i \Delta x_i = -\tfrac{1}{2} \sum_{i=1}^{j} ((\Delta x_{i+1} - \Delta x_i)a - \Delta x_{i+\frac{1}{2}}|a|)(u_{x_{i+1}} - u_{x_i}).$$

For u sufficiently smooth,

$$u_{x_{i+1}} - u_{x_i} = u_{xxi+\frac{1}{2}} \Delta x_{i+\frac{1}{2}} + 0\Delta x^2,$$

hence (A.5) is of $O(\Delta x)$.

Using the initial data at time $t = 0$ $(n = 0)$, $u_i^0 = u(x_i, 0)$ or

$$\text{(A.6)} \qquad e_i^0 = 0.$$

From (A.4), (A.5), and (A.6) it follows that

$$\text{(A.7)} \qquad \sum_{i=1}^{j} e_i^1 \frac{\Delta x_i}{\Delta t} = O(\Delta x).$$

Since (A.7) is true for any j, then

$$\text{(A.8)} \qquad e_i^1 \frac{\Delta x_i}{\Delta t} = O(\Delta x) \quad \forall\, i$$

and the CFL condition (3.21) ensures that

$$e_i^1 = O(\Delta x).$$

By induction it follows from (A.4) that

$$\text{(A.9)} \qquad e_i^{n+1} = O(\Delta x) \quad \forall\, i, n,$$

hence the solution converges with $O(\Delta x)$.

Observations. The same analysis can be applied to the higher-order scheme to show first-order convergence. For example, the Lax–Wendroff scheme used here (3.6), (3.7) has a leading truncation error of

$$\text{(A.10)} \qquad \frac{(\Delta x_{i+\frac{1}{2}} + \Delta x_{i-\frac{1}{2}} - 2\Delta x_i)}{2\Delta x_i} a u_{x_i} + O(\Delta x, \Delta t),$$

which is due to approximating the physical flux component of (3.7) by

$$\text{(A.11)} \qquad \frac{f_i + f_{i+1}}{2}.$$

Second-order convergence can be recovered in two ways: (i) Replace (A.11) by the second-order approximation

$$\text{(A.12)} \qquad \frac{\Delta x_{i+1} f_i + \Delta x_i f_{i+1}}{(\Delta x_{i+1} + \Delta x_i)},$$

which would reduce the truncation error to $O(\Delta x)$. However, the introduction of (A.12) into (3.7) results in a scheme which cannot be shown to be TVD, and the exact shock resolution property of the Roe scheme is lost on an arbitrary nonuniform grid.

(ii) Remove the $O(1)$ truncation error in (A.10) by defining Δx_i to be the mean cell length

$$\Delta x_i = \tfrac{1}{2}\left(\Delta x_{i+\frac{1}{2}} + \Delta x_{i-\frac{1}{2}}\right).$$

This definition effectively removes the distinction between cell-centered and node-based (cell vertex) schemes. However, it is not practical to carry over into two dimensions for the grids used here. (Note that the scheme (3.6), (3.7) would, therefore, achieve second-order convergence if it had been applied using cell vertices as opposed to cell centers in one dimension.)

Appendix B. The Roe decomposition for the Euler equations in two dimensions. In order to satisfy (6.3b), we follow a similar procedure to that presented by Roe and Pike [31] for the one-dimensional case and by Baines [32] in two dimensions. For the Euler equations (§10) the matrix $\boldsymbol{A}(\boldsymbol{u})$ defined in (6.3a) takes the form

$$\begin{bmatrix} 0 & c & s & 0 \\ \frac{1}{2}(\gamma-1)q^2c - uU & U + (2-\gamma)uc & (2-\gamma)vc - V & (\gamma-1)c \\ \frac{1}{2}(\gamma-1)q^2s - vU & V + (2-\gamma)us & U + (2-\gamma)vs & (\gamma-1)s \\ U\left((\gamma-1)q^2 - \frac{\gamma E}{\rho}\right) & \frac{\gamma E}{\rho}c + (1-\gamma)\left(\frac{q^2}{2}c + Uu\right) & \frac{\gamma E s}{\rho} + (1-\gamma)\left(\frac{q^2 s}{2} + Uv\right) & \gamma U \end{bmatrix}, \tag{B.1}$$

and has eigenvectors given by

$$\begin{aligned} e_{1,2} &= [1,\ u \pm ac,\ v \pm as,\ H \pm Ua]^T, \\ e_3 &= [0,\ as,\ -ac,\ -aV]^T, \\ e_4 &= [1,\ u,\ v,\ q^2/2]^T, \end{aligned} \tag{B.2}$$

where H is the enthalpy $(H = q^2/2 + \gamma p/\rho(\gamma-1))$, a is the sound speed $(a = \sqrt{\gamma p/\rho})$,

$$\begin{aligned} U &= \ \ uc + vs, \\ V &= -us + vc, \end{aligned} \tag{B.3}$$

with corresponding eigenvalues

$$\begin{aligned} \lambda_{1,2} &= U \pm a, \\ \lambda_3 &= U, \\ \lambda_4 &= U. \end{aligned} \tag{B.4}$$

Following [31] and [32], α_j (increments of the characteristic variables with respect to the primitive variables (ρ, u, v, p)) are found to be

$$\begin{aligned}\alpha_1 &= [\Delta p + a\rho(c\Delta u + s\Delta v)]/2a^2,\\ \alpha_2 &= [\Delta p - a\rho(c\Delta u + s\Delta v)]/2a^2,\\ \alpha_3 &= \rho(s\Delta u - c\Delta v)/a,\\ \alpha_4 &= \Delta\rho - \Delta p/a^2.\end{aligned} \tag{B.5}$$

Equation (6.3b) is satisfied provided that u, v, h, ρ, and a appearing in (B.1)–(B.5) are defined by

$$\begin{aligned}u &= \frac{u_R\sqrt{\rho_R} + u_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}},\\ v &= \frac{v_R\sqrt{\rho_R} + v_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}},\\ H &= \frac{H_R\sqrt{\rho_R} + H_L\sqrt{\rho_L}}{\sqrt{\rho_R} + \sqrt{\rho_L}},\\ \rho &= \sqrt{\rho_R}\sqrt{\rho_L},\\ a &= \sqrt{(H - q^2/2)(\gamma - 1)},\end{aligned} \tag{B.6}$$

respectively, and the difference operator is defined such that

$$\Delta\rho = \rho_R - \rho_L. \tag{B.7}$$

Acknowledgments. Thanks are given to Professor Earl Thornton for bringing [29] to our attention, Olivier Hardy of the TICOM Computing Laboratory who assisted with some computing difficulties, and to Jan Shrode for her dedication (and patience) in the typing of this report. Thanks are also due to the referees for their helpful comments and criticisms and the references [22], [30].

REFERENCES

[1] P. L. Roe, *Approximate Riemann solvers, parameter vectors and difference schemes*, J. Comput. Phys., 43 (1981), pp. 357–372.

[2] L. Demkowicz and J. T. Oden, *A review of local mesh refinement techniques and corresponding data structures in h-type adaptive finite element methods*, TICOM Report 88-02, Texas Institute for Computational Mechanics, Univ. of Texas, Austin, TX.

[3] A. Harten, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), pp. 357–393.

[4] P. Sweby, *High resolution schemes using flux limiters for hyperbolic conservation laws*, SIAM J. Numer. Anal., 21 (1984), pp. 995–1011.

[5] A. Harten and J. M. Hyman, *Self adjusting grid methods for one-dimensional hyperbolic conservation laws*, J. Comput. Phys., 50 (1983), pp. 235–269.

[6] J. B. Goodman and R. J. LeVeque, *On the accuracy of stable schemes for* 2D *scalar conservation laws*, Math. Comp., 45 (1985), pp. 15–21.

[7] H. C. Yee, *Upwind and symmetric shock capturing schemes*, NASA Technical Memorandum, 89464, NASA Ames Research Center, Moffet Field, CA, 1987.

[8] P. Woodward and P. Colella, *Review article—The numerical simulation of two-dimensional fluid flow with strong shocks*, J. Comput. Phys, 54 (1984), pp. 115–173.

[9] J. T. ODEN, T. STROUBOULIS, AND P. DEVLOO, *Adaptive finite element methods for the analysis of inviscid compressible flow: Part I—Fast refinement, unrefinement and moving mesh methods for unstructured meshes*, Comput. Methods Appl. Mech. Engrg., 59 (1986), pp. 327–362.

[10] R. LOHNER, K. MORGAN, J. PERAIRE, O. C. ZIENKIEWICZ, AND L. KONG, *Finite element methods for compressible flows, in Numerical methods for fluid dynamics* II, K. W. Morton and M. J. Baines, eds., Proc. Conf. Numerical Methods Fluid Dynamics, Academic Press, NY, 1985.

[11] M. J. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.

[12] K. MILLER AND R. N. MILLER, *Moving finite elements part* 1, SIAM J. Numer. Anal., 18 (1981), pp. 1019–1031.

[13] J. N. BRACKBILL AND J. S. SALTZMAN, *Adaptive zoning for singular problems in two dimensions*, J. Comput. Phys., 46 (1982), pp. 342–368.

[14] P. L. ROE, *Some contributions to the modelling of discontinuous flows*, Lectures in Applied Mathematics, Volume 22, Large Scale Computations in Fluid Mechanics, S. Osher, B. Engquist, and R. Somerville, eds., 1985, pp. 163–193.

[15] ———, *The use of the Riemann problem in finite difference schemes*, Proc. Seventh Internat. Conf. Numerical Methods Fluid Dynamics, Stanford University, Stanford, CA, 1980. Also in Lecture Notes in Phys., 141, Springer-Verlag, Berlin, New York, 1981, pp. 354–359.

[16] B. VAN LEER, *Towards the ultimate conservative difference scheme* II. *Monotonicity and conservation combined in a second order scheme*, J. Comput. Phys., 14 (1974), pp. 361–370.

[17] S. CHAKRAVARTHY AND S. OSHER, *High resolution applications of the Osher upwind scheme for the Euler equations*, AIAA Paper, 6th Computational Fluid Dynamics Conference, 1983.

[18] R. SANDERS, *On convergence of monotone finite difference schemes with variable space differencing*, Math. Comp., 40 (1983), pp. 91–106.

[19] P. SWEBY AND M. J. BAINES, *On convergence of Roe's scheme for the general non-linear scalar wave equation*, J. Comput. Phys., 56 (1984), pp. 135–148.

[20] A. Y. LEROUX, *Convergence of an accurate scheme for first order quasi-linear equations*, RAIRO Anal. Numer., 15 (1981), pp. 151–170.

[21] J. PIKE, *Grid adaptive algorithms for the solution of the Euler equations on irregular grids*, J. Comput. Phys., 71 (1987), pp. 194–223.

[22] H. O. KREISS, T. A. MANTEUFFEL, B. SWARTZ, B. WENDROFF, AND A. B. WHITE, JR., *Supra-convergent schemes on irregular grids*, Math. Comp., 47 (1986), pp. 537–554.

[23] G. STRANG, *On the construction and comparison of finite difference schemes*, SIAM J. Numer. Anal., 5 (1968), pp. 506–517.

[24] L. DEMKOWICZ, *An adaptive h-p finite element strategy for two-dimensional boundary-value problems—A theory manual*, TICOM Report, Texas Institute for Computational Mechanics, Univ. of Texas, Austin, TX, 1988.

[25] P. H. DEVLOO, *An h-p adaptive finite element method for steady compressible flow*, Ph.D. thesis, The University of Texas, Austin, TX, 1987.

[26] M. G. EDWARDS, *Moving element methods with emphasis on the Euler equations*, Ph.D. thesis, Math Dept., Reading University, U.K., 1987.

[27] R. COURANT AND K. O. FRIEDRICHS, *Supersonic flow and shock waves*, Applied Mathematical Sciences, Vol. 21, Springer-Verlag, Berlin, New York.

[28] K. S. BEY, E. A. THORNTON, P. DECHAUNPAI, AND R. RAMAKRISHNAN, *A new finite element approach for prediction of aerothermal loads—progress in inviscid flow computations*, AIAA Paper 85-1533-CP, 1985.

[29] F. BILLIG, *Shock wave shapes around spherical and cylindrical-nosed bodies*, J. Spacecraft, 4 (1967), pp. 822–823.

[30] M. J. BERGER, *On conservation at grid interfaces*, SIAM J. Numer. Anal., 24 (1987), pp. 967–994.

[31] P. L. ROE AND J. PIKE, *Efficient construction and utilization of approximate Riemann solutions*, Comput. Methods Appl. Sci. Engrg., VI (1984), pp. 449–518.

[32] M. J. BAINES, *Two-dimensional shock recognition and Roe's scheme*, Numerical Analysis Report 10/83, Reading University, U.K., 1983.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 218–230, January 1993

INTERACTION SPLINES WITH REGULAR DATA: AUTOMATICALLY SMOOTHING DIGITAL IMAGES*

CHONG GU†

Abstract. The idea of interaction spline was first proposed by Barry [*Ann. Statist.*, 14 (1986), pp. 934–953] in the context of nonparametric Bayesian multivariate regression. Wahba [*Computer Science and Statistics*, American Statistical Association, 1986, pp. 75–80] shows that Barry's model is a special case of a wider class of smooth functions which can be constructed from the tensor product of reproducing kernel Hilbert spaces. Algorithms for computing interaction splines with arbitrarily scattered data have been developed by Gu, Bates, Chen, and Wahba [*SIAM J. Matrix Anal. Appl.*, 10 (1989), pp. 457–480] and Gu and Wahba [*SIAM J. Sci. Statist. Comput.*, 12 (1991), pp. 383–398]. These algorithms require $O(n^3)$ flops for sample size n. The purpose of this work is to develop fast algorithms for computing interaction splines with data sampled from a regular product mesh. The algorithm is of order $O(n)$ for practical problem size. Smoothing parameters can be tuned using the generalized cross-validation. A possible application of the technique is image smoothing. In image smoothing, a periodic interaction spline is observed to be a low-pass filter of a certain form. Simulated examples are presented and practical aspects of the algorithms are discussed.

Key words. adaptive filter, conjugate gradient iteration, fast Fourier transform, generalized cross-validation, image smoothing, smoothing parameter

AMS(MOS) subject classifications. 65D07, 65D10, 65F10, 65K10, 65U05

1. Introduction.

Suppose one observes $y_j = f(\mathbf{x}_j) + \epsilon_j$, $j = 1, \ldots, n$, where $\mathbf{x}_j \in [0,1]^d$, $\mathbf{E}\epsilon_j = 0$, $\text{var}(\epsilon_j) \propto w_j^{-1}$, and the ϵ_j's are uncorrelated. The solution f_λ to the penalized least square problem

$$\min \frac{1}{n}\sum_{j=1}^{n} w_j(y_j - f(\mathbf{x}_j))^2 + \lambda\|P_1 f\|^2, \qquad \text{s.t. } f \in \mathcal{H}, \tag{1.1}$$

defines a smoothing spline, where $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS) of functions on domain $\mathcal{X}$ with norm $\|\cdot\|$, and P_1 is the orthogonal projector onto a subspace $\mathcal{H}_1$ of co-dimension $M < n$. The solution has an expression $f_\lambda = \sum_{\nu=1}^{M} d_\nu \phi_\nu + \sum_{j=1}^{n} c_j \xi_j$, where $\{\phi_\nu\}_{\nu=1}^{M}$ is a basis for $\mathcal{H}_0$, the orthogonal complement of $\mathcal{H}_1$, and $\xi_j = P_1\tilde{\xi}_j$, where $\tilde{\xi}_j$ is the representer of the evaluation functional $[\mathbf{x}_j](\cdot)$. The reproducing kernel (RK) of $\mathcal{H}$ is a bivariate function $R(\cdot,\cdot)$ (symmetric in its two slots) such that $R(\mathbf{x},\cdot)$ is the representer of $[\mathbf{x}](\cdot)$. For $\mathcal{H} = \mathcal{H}_0 \oplus \mathcal{H}_1$, $\mathcal{H}_0$ and $\mathcal{H}_1$ are also RKHSs and the sum of their RKs R_0 and R_1 is equal to the RK R of $\mathcal{H}$. Substituting the solution expression into (1.1), the problem reduces to solving

$$\min \frac{1}{n}(\mathbf{y} - Q\mathbf{c} - S\mathbf{d})^T W(\mathbf{y} - Q\mathbf{c} - S\mathbf{d}) + \lambda\mathbf{c}^T Q\mathbf{c}, \tag{1.2}$$

where $(Q)_{j,k} = R_1(\mathbf{x}_j, \mathbf{x}_k)$, $(S)_{j,\nu} = \phi_\nu(\mathbf{x}_j)$, and $W = \text{diag}(w_1, \ldots, w_n)$; see [17, §1.3]. See also [1] for the definition and properties of RKHS. Note that the RK is uniquely determined by the norm and vice versa. The notion of smoothness for $f \in \mathcal{H}$ is defined as the deviation of f from $\mathcal{H}_0$, namely $\|P_1 f\|^2$. It is possible to specify different RKs for the same set of functions to reflect different meanings by the word "smoothness."

*Received by the editors May 9, 1990; accepted for publication (in revised form) January 22, 1992. This research was supported by the Natural Sciences and Engineering Research Council of Canada while the author was at the University of British Columbia. The revision was supported by National Science Foundation grant DMS-9101730.

†Department of Statistics, Purdue University, West Lafayette, Indiana 47907.

Interaction splines are defined on the tensor product RKHS (cf. [1]). Consider RKHS $\mathcal{H}^1 = \mathcal{H}_0^1 \oplus \mathcal{H}_1^1$ with the RK $R^1 = R_0^1 + R_1^1$ on $[0,1]$ and $\mathcal{H}^2 = \mathcal{H}_0^2 \oplus \mathcal{H}_1^2$ with the RK $R^2 = R_0^2 + R_1^2$ on $[0,1]$, where $\mathcal{H}_0^1$ and $\mathcal{H}_0^2$ are of finite dimension. The tensor product RKHS on $[0,1]^2$ has a tensor sum decomposition $\mathcal{H} = \mathcal{H}^1 \otimes \mathcal{H}^2 = (\mathcal{H}_0^1 \otimes \mathcal{H}_0^2) \oplus (\mathcal{H}_1^1 \otimes \mathcal{H}_0^2) \oplus (\mathcal{H}_0^1 \otimes \mathcal{H}_1^2) \oplus (\mathcal{H}_1^1 \otimes \mathcal{H}_1^2) = \mathcal{H}_{0,0} \oplus \mathcal{H}_{1,0} \oplus \mathcal{H}_{0,1} \oplus \mathcal{H}_{1,1}$. The norm in tensor product RKHS is usually defined indirectly via its RK, which is often taken as the product of the RKs of the component spaces; see [1]. For example, a RK in $\mathcal{H}_{1,0}$ can be taken as $R_{1,0}((x_1,x_2),(x_1',x_2')) = R_1^1(x_1,x_1')R_0^2(x_2,x_2')$. Let $\|\cdot\|$ be the norm in the tensor product RKHS $\mathcal{H}$ corresponding to the RK R^1R^2. It can be shown that $\|f\|_\theta^2 = \theta_0^{-1}\|P_{0,0}f\|^2 + \theta_1^{-1}\|P_{1,0}f\|^2 + \theta_2^{-1}\|P_{0,1}f\|^2 + \theta_{1,2}^{-1}\|P_{1,1}f\|^2$ defines a class of norms with the corresponding RK $\theta_0 R_0^1R_0^2 + \theta_1 R_1^1R_0^2 + \theta_2 R_0^1R_1^2 + \theta_{1,2}R_1^1R_1^2$, where the θ's are all positive and the $P_{i,j}$'s are projectors onto $\mathcal{H}_{i,j}$'s (cf. [11]). Letting $\mathcal{H}_0 = \mathcal{H}_{0,0}$ and $\mathcal{H}_1 = \mathcal{H}_{1,0} \oplus \mathcal{H}_{0,1} \oplus \mathcal{H}_{1,1}$ in (1.1), with the norm indexed by θ, one obtains interaction splines with multiple smoothing parameters λ/θ_1, λ/θ_2, and $\lambda/\theta_{1,2}$, where the θ's determine the contributions (weights) of the norms of the individual spaces $\mathcal{H}_{1,0}$, $\mathcal{H}_{0,1}$, and $\mathcal{H}_{1,1}$ to the combined norm of the space $\mathcal{H}_1$; see, e.g., [10], [11], [9], and [3]. Let $\{\phi_\nu^1\}_{\nu=1}^{M_1}$ be a basis in $\mathcal{H}_0^1$ and $\{\phi_\nu^2\}_{\nu=1}^{M_2}$ a basis in $\mathcal{H}_0^2$. It can be seen that the $M = M_1M_2$ product functions $\phi_{\nu_1}^1\phi_{\nu_2}^2$ form a basis for $\mathcal{H}_{0,0}$. The quantities in (1.2) are thus

$$(1.3)\qquad (Q)_{j,k} = \theta_1(R_1^1R_0^2)(\mathbf{x}_j,\mathbf{x}_k) + \theta_2(R_0^1R_1^2)(\mathbf{x}_j,\mathbf{x}_k) + \theta_{1,2}(R_1^1R_1^2)(\mathbf{x}_j,\mathbf{x}_k),$$

$$(1.4)\qquad (S)_{j,\nu} = \phi_\nu(\mathbf{x}_j) = (\phi_{\nu(1)}^1\phi_{\nu(2)}^2)(\mathbf{x}_j).$$

The generalization of the above procedure to $(d > 2)$-dimensional space is straightforward; see, e.g., [11]. Details are omitted here.

A class of RKHS of univariate functions on $[0,1]$ is the space $W_2^m = \{f : f^{(\nu)}$ abs.cont., $\nu = 0,\dots,m-1, \int (f^{(m)})^2 < 0\}$ with the norm

$$\|f\|^2 = \sum_{\nu=0}^{m-1}\left(\int_0^1 f^{(\nu)}\right)^2 + \int_0^1 (f^{(m)})^2$$

and the corresponding RK

$$R(s,t) = \sum_{\nu=0}^{m} k_\nu(s)k_\nu(t) + (-1)^{m-1}k_{2m}(|s-t|),$$

where $k_\nu = B_\nu/\nu!$ and B_ν is the νth Bernoulli polynomial. Write $W_2^m = \pi_m \oplus \mathcal{S}_m$, where π_m is the space of polynomials of order less than m with the norm $\sum_{\nu=0}^{m-1}(\int_0^1 f^{(\nu)})^2$, and $\mathcal{S}_m$ contains functions satisfying $\int f^{(\nu)} = 0$, $\nu = 0,\dots,m-1$, with the norm $\int_0^1(f^{(m)})^2$. The RK of π_m is

$$(1.5)\qquad \sum_{\nu=0}^{m-1} k_\nu(s)k_\nu(t),$$

and the RK of $\mathcal{S}_m$ is

$$(1.6)\qquad k_m(s)k_m(t) + (-1)^{m-1}k_{2m}(|s-t|);$$

see [4]. The subset of periodic functions of W_2^m comprises a RKHS of the form $\{1\} \oplus \mathcal{S}_m^{per}$, where $\mathcal{S}_m^{per}$ is the periodic subset of $\mathcal{S}_m$. The RK of $\{1\}$ is 1 and the RK of $\mathcal{S}_m^{per}$ is $(-1)^{m-1}k_{2m}(|s-t|)$; see [17, §2.1]. In fact, $\{k_\nu\}_{\nu=0}^{m-1}$ is an orthonormal basis for π_m with norm $\sum_{\nu=0}^{m-1}(\int f^{(\nu)})^2$, and k_ν's individually span m mutually orthogonal one-dimensional subspaces of π_m with norms $(\int f^{(\nu)})^2$. We shall use $\phi_\nu = k_{\nu-1}$, $\nu = 1, \dots, m$, as a basis for π_m. In this article, the term interaction spline refers to the case where the component spaces are either W_2^m's or their periodic restrictions.

The idea of interaction spline was first proposed by Barry [2] in the context of nonparametric Bayesian regression. Barry's model is equivalent to using $W_2^1 = \{1\} \oplus \mathcal{S}_1$ for all of the component spaces. The formulation sketched above follows Wahba's development [16]. The computation of interaction splines for arbitrarily scattered $\mathbf{x}_j$'s is studied by Gu, Bates, Chen, and Wahba [10] and Gu and Wahba [11], where it is argued that $O(n^3)$ operations are inevitable. Some asymptotic properties of interaction splines were investigated by Chen [3]. The purpose of this article is to develop fast algorithms for computing interaction splines, assuming that the $\mathbf{x}_j$'s fall on a tensor product mesh of equally spaced grids. We will restrict our development to $d = 2$. Extension to $d > 2$ is straightforward but tedious. A possible application of the technique is the automatic smoothing of digital images.

In §2, basic algorithms are developed for situations of different complexities. Section 3 discusses the implication of the periodic splines as automatic low-pass filters. In §4, numerical efficiency of the building blocks is studied and examples are presented. Section 5 concludes the article with discussion.

2. Algorithms.

We assume $d = 2$. The operator "$\otimes$" will hereafter stand for the Kronecker product of matrices. The superscript "H" will stand for the conjugate transpose of matrices and vectors.

2.1. Preliminaries.

It can be shown that the linear system

$$(WQ + n\lambda I)\mathbf{c} + WS\mathbf{d} = W\mathbf{y}, \quad S^T\mathbf{c} = 0 \tag{2.1}$$

gives a solution to problem (1.2); see, e.g., [9]. We will use the generalized cross-validation (GCV) method of Craven and Wahba [4] to automatically choose the smoothing parameters, which are the $n\lambda$ and the θ's hidden in the matrix Q. The GCV method has been proved to be asymptotically optimal for minimizing prediction mean square errors; see [17] for more references.

For $W = I$, letting $\hat{\mathbf{y}} = A(\lambda, \theta)\mathbf{y}$ denote the vector of predictions $f_\lambda(\mathbf{x}_j)$, the GCV score to be minimized is defined by $V(\lambda, \theta) = n(\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}})/[\text{tr}(I - A(\lambda, \theta))]^2$. It can be shown that $(n\lambda)\mathbf{c} = \mathbf{y} - \hat{\mathbf{y}}$ and hence $V = n\mathbf{c}^T\mathbf{c}/[\text{tr}B]^2$ where $\mathbf{c} = B\mathbf{y}$. Let $P_S^\perp = I - P_S = I - S(S^TS)^{-1}S^T$ be the projection matrix to the orthogonal complement of the column space of S. Since $S^T\mathbf{c} = 0$, it is easy to see that $\mathbf{c} = P_S^\perp\mathbf{c}$. Applying $P_S^\perp$ from the left to the first equation of (2.1) (with $W = I$), it follows that

$$(P_S^\perp Q P_S^\perp + n\lambda I)\mathbf{c} = P_S^\perp\mathbf{y}. \tag{2.2}$$

We will solve $\mathbf{c}$ from (2.2) and calculate V via

$$V = n\mathbf{c}^T\mathbf{c}/[\text{tr}(P_S^\perp Q P_S^\perp + n\lambda I)^{-1}P_S^\perp]^2. \tag{2.3}$$

$\mathbf{d}$ can be solved from the first equation of (2.1). The computation for a general W will be discussed in §2.4.

Suppose one has $n = n_1n_2$ data on the regular mesh $\{(x_{1,j_1}, x_{2,j_2}) : j_1 = 1, \dots, n_1;$ $j_2 = 1, \dots, n_2\}$, where $x_{1,j} = (j - .5)/n_1$, $x_{2,j} = (j - .5)/n_2$. Define S_1 and S_2 with entries $(S_1)_{j,\nu} = \phi_\nu(x_{1,j})$ and $(S_2)_{j,\nu} = \phi_\nu(x_{2,j})$, where, $\phi_\nu = k_{\nu-1}$ form a basis for $\mathcal{H}_0^1$ and $\mathcal{H}_0^2$. From (1.5), it is easy to see that $R_0^1(x_{1,j}, x_{1,k}) = (S_1S_1^T)_{j,k}$ and $R_0^2(x_{2,j}, x_{2,k}) = (S_2S_2^T)_{j,k}$. Also define Q_1 and Q_2 such that $(Q_1)_{j,k} = R_1^1(x_{1,j}, x_{1,k})$ and $(Q_2)_{j,k} = R_1^2(x_{2,j}, x_{2,k})$. Order the data such that j_2 varies most frequently and order the basis functions of $\mathcal{H}_{0,0}$, $\phi_\nu = \phi^1_{\nu(1)}\phi^2_{\nu(2)}$, such that $\nu(2)$ varies most frequently. It follows from (1.3) and (1.4) that

$$\begin{aligned} Q &= \theta_{1,2}(Q_1 \otimes Q_2) + \theta_1(Q_1 \otimes S_2S_2^T) + \theta_2(S_1S_1^T \otimes Q_2), \\ S &= S_1 \otimes S_2. \end{aligned} \tag{2.4}$$

The developments below are based on (2.4).

2.2. Periodic spline with equal weights. Defining P_n as an $n \times n$ matrix with the (i,j)th entry $(P_n)_{i,j} = (-1)^{m-1}k_{2m}(|i-j|/n)$, where $k_{2m} = B_{2m}/(2m)!$ is the scaled Bernoulli polynomial, the key to the developments is the spectral decomposition

$$\Gamma_n^H P_n \Gamma_n = \Lambda_n,$$

where Γ_n is the Fourier matrix with entries $(\Gamma_n)_{j,k} = n^{-1/2}\exp(2\pi \mathrm{i}(j-1)(k-1)/n)$, where $\mathrm{i} = \sqrt{-1}$, and $\Lambda_n = \operatorname{diag}(\lambda_{1,n}, \ldots, \lambda_{n,n})$, where

$$\begin{aligned} \lambda_{1,n} &= 2n(2\pi)^{-2m}\textstyle\sum_{j=1}^{\infty}(jn)^{-2m}, \\ \lambda_{\nu,n} &= n(2\pi)^{-2m}\textstyle\sum_{j=-\infty}^{\infty}(\nu - 1 + jn)^{-2m}, \qquad \nu = 2, \ldots, n; \end{aligned} \tag{2.5}$$

see [4]. If both $\mathcal{H}^1$ and $\mathcal{H}^2$ are periodic, then $S_1 = 1$, $S_2 = 1$, $\Gamma_{n_1}^H Q_1 \Gamma_{n_1} = \Lambda_{n_1} = \Lambda_1$, and $\Gamma_{n_2}^H Q_2 \Gamma_{n_2} = \Lambda_{n_2} = \Lambda_2$, where S_1 has length n_1; S_2 has length n_2; Γ_{n_1}, Q_1, and Λ_1 are of dimension $n_1 \times n_1$; and Γ_{n_2}, Q_2, and Λ_2 have dimension $n_2 \times n_2$. Writing $\Gamma = \Gamma_{n_1} \otimes \Gamma_{n_2}$, $\tilde{\mathbf{y}} = \Gamma^H \mathbf{y}$, $\tilde{S} = \Gamma^H S = (\Gamma_{n_1}^H S_1) \otimes (\Gamma_{n_2}^H S_2)$, $\tilde{Q} = \Gamma^H Q \Gamma$, and $\tilde{\mathbf{c}} = \Gamma^H \mathbf{c}$, (2.2) becomes

$$(P_{\tilde{S}}^\perp \tilde{Q} P_{\tilde{S}}^\perp + n\lambda I)\tilde{\mathbf{c}} = P_{\tilde{S}}^\perp \tilde{\mathbf{y}}, \tag{2.6}$$

where $P_{\tilde{S}}^\perp = I - P_{\tilde{S}} = I - P_{\tilde{S}_1} \otimes P_{\tilde{S}_2}$. It follows that $\tilde{S}_1 = \sqrt{n_1}\mathbf{e}_1$ and $\tilde{S}_2 = \sqrt{n_2}\mathbf{e}_1$, where $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$ have lengths n_1 and n_2, respectively. It can be seen that $P_{\tilde{S}}^\perp = \operatorname{diag}(0, I)$, $\tilde{Q} = \theta_{1,2}(\Lambda_1 \otimes \Lambda_2) + \theta_1(\Lambda_1 \otimes J_2) + \theta_2(J_1 \otimes \Lambda_2)$, where

$$J_1 = \operatorname{diag}(n_1, O)_{n_1 \times n_1} \quad \text{and} \quad J_2 = \operatorname{diag}(n_2, O)_{n_2 \times n_2}. \tag{2.7}$$

Hence (2.6) is a diagonal system with real diagonals. Since

$$\mathbf{c}^T\mathbf{c} = \tilde{\mathbf{c}}^H\tilde{\mathbf{c}} = \tilde{\mathbf{y}}^H P_{\tilde{S}}^\perp (P_{\tilde{S}}^\perp \tilde{Q} P_{\tilde{S}}^\perp + n\lambda I)^{-2} P_{\tilde{S}}^\perp \tilde{\mathbf{y}}$$

and

$$\operatorname{tr}B = \operatorname{tr}(\Gamma^H B \Gamma) = \operatorname{tr}[(P_{\tilde{S}}^\perp \tilde{Q} P_{\tilde{S}}^\perp + n\lambda I)^{-1} P_{\tilde{S}}^\perp],$$

the score V has an expression explicit in λ's and θ's and so do its derivatives. Hence V can be minimized with respect to the smoothing parameters via an appropriate Newton-type method. There is one redundant dimension in the smoothing parameters and one can simply fix $n\lambda$. See, e.g., [11], for a related algorithm. The calculation of $\tilde{\mathbf{y}} = \Gamma^H \mathbf{y}$, which is the forward two-dimensional discrete Fourier transform (DFT) of $\mathbf{y}$ on a two-dimensional array, can be conducted via the fast Fourier transform (FFT). A backward DFT of $\mathbf{c} = \Gamma\tilde{\mathbf{c}}$ via FFT gives the solution to the original problem (2.2). We can check that the one-dimensional coefficient d in this setup is equal to the overall average of the y's, which is independent of the smoothing parameters.

2.3. General spline with equal weights. For a general problem, $\tilde{Q}$ and $P_{\tilde{S}}$ are not diagonal. However, the matrix–vector multiplication of $\tilde{Q}\mathbf{x}$ (and of $P_{\tilde{S}}\mathbf{x}$) is of order $O(n)$ (note that $Q\mathbf{x}$ is not), and hence for fixed $n\lambda$'s and θ's one can solve (2.6) via the preconditioned conjugate gradient iteration; see, e.g., [8, Chap. 10]. To be specific, note that $\tilde{Q}_i = \Lambda_i + \tilde{\mathbf{t}}_i\tilde{\mathbf{t}}_i^H$ where $\tilde{\mathbf{t}}_i = \Gamma^H\mathbf{t}_i$ and $\mathbf{t}_i = (k_m(x_{i,1}), \ldots, k_m(x_{i,n_i}))^T$, $i = 1, 2$ (cf. (1.6)). It can be shown that

$$P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} = P_{\tilde{S}}^{\perp}(\theta_{1,2}(\Lambda_1 + \tilde{\mathbf{t}}_1\tilde{\mathbf{t}}_1^H + (\theta_2/\theta_{1,2})\tilde{S}_1\tilde{S}_1^H) \otimes (\Lambda_2 + \tilde{\mathbf{t}}_2\tilde{\mathbf{t}}_2^H + (\theta_1/\theta_{1,2})\tilde{S}_2\tilde{S}_2^H))P_{\tilde{S}}^{\perp}.$$

A convenient preconditioner for the conjugate gradient iteration is

$$C = \theta_{1,2}(\Lambda_1 \otimes \Lambda_2) + \theta_1(\Lambda_1 \otimes J_2) + \theta_2(J_1 \otimes \Lambda_2) + n\lambda I, \tag{2.8}$$

where J_1 and J_2 are given in (2.7). It is easily seen that $P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} + n\lambda I - C$ is of lower rank. We assume that S_1 and S_2 both have a column of constant.

To evaluate the GCV score V, one needs to calculate $\text{tr}((P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} + n\lambda I)^{-1}P_{\tilde{S}}^{\perp})$. Girard [6] noticed that if one replaces $\mathbf{y}$ by a vector $\boldsymbol{\epsilon}$ of independent and identically distributed (i.i.d.) random noise with mean 0 and variance 1 and solves for $\mathbf{c}_\epsilon$, then $\mathbf{E}\mathbf{c}_\epsilon^T\boldsymbol{\epsilon} = \text{tr}((P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} + n\lambda I)^{-1}P_{\tilde{S}}^{\perp})$. Therefore, he suggests using $\mathbf{c}_\epsilon^T\boldsymbol{\epsilon}$ or a slightly better version $n\mathbf{c}_\epsilon^T\boldsymbol{\epsilon}/\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$ to estimate the trace. Simulation results about this method can be found in [6] and further theoretical justifications can be found in [6] and [7].

We now describe another approximation for the trace term in the denominator of the GCV score which is appropriate for the current setup. Let $\mu_1 = \cdots = \mu_M = 0 < \cdots \leq \mu_n$ be the ordered eigenvalues of $P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp}$, where M is the number of columns in S. Write $P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} = U\Delta U^H = (U_1, U_2)\Delta(U_1, U_2)^H$, where $\Delta = \text{diag}(\mu_j)$, U_1 consists of the eigenvectors of $\mu_1 = \cdots = \mu_M = 0$, and U_2 consists of the eigenvectors of the remaining $n - M$ positive eigenvalues. Since $P_{\tilde{S}}^{\perp}U_1 = O$ and $P_{\tilde{S}}^{\perp}$ is a projector, $P_{\tilde{S}}^{\perp} = U_2U_2^H$. It then follows that $\text{tr}((P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} + n\lambda I)^{-1}P_{\tilde{S}}^{\perp}) = \text{tr}((\Delta + n\lambda I)^{-1}U^HU_2U_2^HU) = \sum_{j=M+1}^{n}(n\lambda + \mu_j)^{-1}$. Writing $\nu_1 \leq \cdots \leq \nu_n$ for the eigenvalues of $\tilde{Q}/\theta_{1,2} = (\Lambda_1 + \tilde{\mathbf{t}}_1\tilde{\mathbf{t}}_1^H + (\theta_2/\theta_{1,2})\tilde{S}_1\tilde{S}_1^H) \otimes (\Lambda_2 + \tilde{\mathbf{t}}_2\tilde{\mathbf{t}}_2^H + (\theta_1/\theta_{1,2})\tilde{S}_2\tilde{S}_2^H)$, it can be shown that $\theta_{1,2}\nu_j \leq \mu_{j+M} \leq \theta_{1,2}\nu_{j+M}$, $j = 1, \ldots, n - M$; see, e.g., [14, pp. 316-317, ex. 4]. Hence, the ν_j's can be used to calculate bounds for the denominator of the GCV score. Computationally, the ν_j's are just products of the eigenvalues of $Q_1 + \mathbf{t}_1\mathbf{t}_1^T + (\theta_2/\theta_{1,2})S_1S_1^T$ and $Q_2 + \mathbf{t}_2\mathbf{t}_2^T + (\theta_1/\theta_{1,2})S_2S_2^T$, which are available in $O(n_1^3) + O(n_2^3)$ flops. For the problems covered in this article, this method is preferable to Girard's method in both speed and precision; see §4.

2.4. Unequal weights. For a problem with unequal weights, (2.1) leads to

$$(P_{WS}^{\perp}W^{1/2}QW^{1/2}P_{WS}^{\perp} + n\lambda I)(W^{-1/2}\mathbf{c}) = P_{WS}^{\perp}(W^{1/2}\mathbf{y}), \tag{2.9}$$

where $P_{WS}^{\perp} = I - P_{WS} = I - W^{1/2}S(S^TWS)^{-1}S^TW^{1/2}$. Assuming $W^{1/2}\hat{\mathbf{y}} = A(W^{1/2}\mathbf{y})$ and $W^{-1/2}\mathbf{c} = B(W^{1/2}\mathbf{y})$, the GCV score is defined by

$$V = n(\mathbf{y} - \hat{\mathbf{y}})^TW(\mathbf{y} - \hat{\mathbf{y}})/[\text{tr}(I - A)]^2 = n\mathbf{c}^TW^{-1}\mathbf{c}/[\text{tr}B]^2, \tag{2.10}$$

where the last equation follows the fact that $(n\lambda)(W^{-1/2}\mathbf{c}) = W^{1/2}(\mathbf{y} - \hat{\mathbf{y}})$. The conjugate gradient iteration can be used to solve (2.9) for $W^{-1/2}\mathbf{c}$. It can be shown that

$$\begin{aligned}P^{\perp}_{WS}W^{1/2}QW^{1/2}P^{\perp}_{WS} = P^{\perp}_{WS}W^{1/2}\Gamma \\ &(\theta_{1,2}(\Lambda_1 + \tilde{\mathbf{t}}_1\tilde{\mathbf{t}}_1^H + (\theta_2/\theta_{1,2})\tilde{S}_1\tilde{S}_1^H) \\ &\quad\otimes(\Lambda_2 + \tilde{\mathbf{t}}_2\tilde{\mathbf{t}}_2^H + (\theta_1/\theta_{1,2})\tilde{S}_2\tilde{S}_2^H)) \\ &\Gamma^H W^{1/2}P^{\perp}_{WS}.\end{aligned}$$

We are not able to identify a convenient effective preconditioner for this iteration. A standard conjugate gradient iteration for solving (2.9) needs a pair of FFTs in each step.

The denominator of the GCV score involves

$$\mathrm{tr}((P^{\perp}_{WS}W^{1/2}QW^{1/2}P^{\perp}_{WS} + n\lambda I)^{-1}P^{\perp}_{WS}).$$

As indicated in §2.3, Girard's method may be used to approximate the trace. To apply the trace bounds based on the eigenvalues, one needs $W = W_1 \otimes W_2$, which in general is not true. Nevertheless, one might use an approximation of W with the form $W_1 \otimes W_2$ to obtain approximate bounds. An obvious choice for W_1 and W_2 is $W_1 = \mathrm{diag}(\tilde{w}_{1,\cdot}, \ldots, \tilde{w}_{n_1,\cdot})$ and $W_2 = \mathrm{diag}(\tilde{w}_{\cdot,1}, \ldots, \tilde{w}_{\cdot,n_2})$, where $\tilde{w}_{i,\cdot}$'s and $\tilde{w}_{\cdot,j}$'s are the row-wise and column-wise geometric means of the weight matrix W.

2.5. Penalized likelihood. For categorical and/or asymmetric random schemes it is usually assumed that y_j follow a distribution with log likelihood $l_j(\eta_j) = (y_j\eta_j - b(\eta_j))/a(\psi)$, where $\eta_j = f(\mathbf{x}_j)$ is the parameter to be modeled on $\mathbf{x}_j$ and ψ is a nuisance parameter common to all observations. The penalized likelihood method estimates f via

$$\min\ -\frac{2}{n}\sum_{j=1}^{n} l_j(f(\mathbf{x}_j)) + \lambda\|P_1 f\|^2, \qquad f \in \mathcal{H}.$$

It can be shown that the Newton method iterates on

$$\min\ \frac{1}{n}\sum_{j=1}^{n} w_j(\tilde{\eta}_j - f(\mathbf{x}_j))^2 + \lambda\|P_1 f\|^2, \qquad f \in \mathcal{H},$$

where $\tilde{\eta}_j = \eta_{0j} - u_j/w_j$, $\eta_{0,j}$ is the previous iterate of η_j, $u_j = -dl_j/d\eta_j|_{\eta_{0j}}$, and $w_j = -d^2l_j/d\eta_j^2|_{\eta_{0j}}$; see [9]. See also [12] and [13]. The algorithm of §2.4 can be used to implement the steps of this Newton iteration.

3. Interaction splines as low-pass filters. The algorithm of §2.2 for periodic interaction splines can be viewed as automatic low-pass filters for high-dimensional image. To see this, remember that $\hat{\mathbf{y}} = \mathbf{y} - (n\lambda)\mathbf{c}$, so the two-dimensional DFT of $\hat{\mathbf{y}}$ is

$$\tilde{\hat{\mathbf{y}}} = (I - (n\lambda)(P^{\perp}_{\tilde{S}}\tilde{Q}P^{\perp}_{\tilde{S}} + n\lambda I)^{-1}P^{\perp}_{\tilde{S}})\tilde{\mathbf{y}}.$$

Hence, the smoothed image is obtained by damping the spectrum of the observed image. The damping factor for the (i,j)th cell, $i, j > 1$, is

$$\delta_{i,j} = \theta_{1,2}\lambda_{i,n_1}\lambda_{j,n_2}/(\theta_{1,2}\lambda_{i,n_1}\lambda_{j,n_2} + n\lambda)$$

(cf. (2.5) and (2.6)); the factor for the $(i,1)$st cell, $i > 1$, is $\delta_{i,1} = (\theta_{1,2}\lambda_{i,n_1}\lambda_{1,n_2} + n_2\theta_1\lambda_{i,n_1})/((\theta_{1,2}\lambda_{i,n_1}\lambda_{1,n_2} + n_2\theta_1\lambda_{i,n_1}) + n\lambda)$; the factor for the $(1,j)$th cell, $j > 1$, is

$\delta_{1,j} = (\theta_{1,2}\lambda_{1,n_1}\lambda_{j,n_2} + n_1\theta_2\lambda_{j,n_2})/((\theta_{1,2}\lambda_{1,n_1}\lambda_{j,n_2} + n_1\theta_2\lambda_{j,n_2}) + n\lambda)$; and the factor for the $(1,1)$st cell is 1. The magnitude of $\lambda_{\nu,n}$ declines at a rate $O(\nu^{-2m})$ over $\nu = 2,\ldots,n/2$ (cf. [4]), hence the method is a low-pass filter. The threshold frequencies of the filter are determined by the magnitude of the $n\lambda/\theta$'s relative to that of the $\lambda_{\nu,n}$'s. Larger m leads to a greater declining rate in $\lambda_{\nu,n}$, and in turn to a sharper threshold boundary. The GCV method in this setting can be viewed as an effective method for automatically tuning the threshold frequencies.

As an illustration, we generated a 64×64 image on $(0,1)^2$ by adding Gaussian noise with mean 0 and variance .04 to the function

$$[\cos(3\pi((x_1 - .5)^2 + (x_2 - .5)^2)) + \cos(2\pi(x_1 - .5))\sin(2\pi(x_2 - .5))]/3.2, \tag{3.1}$$

and applied the algorithm to the image with $m = 2$. See §4 for more details about this example. The frequency responses and the equivalent smoothing kernel of the GCV tuned filter are plotted in Fig. 3.1.

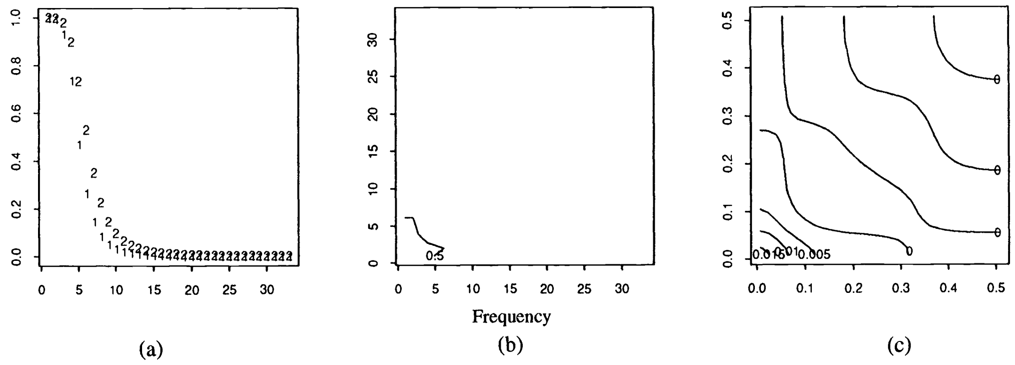


FIG. 3.1. *An automatic low-pass filter.* (a) *Frequency response on the edges: "1" indicates $\delta_{i,1}$ and "2" indicates $\delta_{1,j}$* ; (b) *Half-power contour of $\delta_{i,j}$* ; (c) *Contour of a quarter of the equivalent smoothing kernel on the space domain.*

4. Examples and practicalities. The algorithms described in §2 are certain combinations of primitive building blocks, namely the FFT, the Newton iteration, the preconditioned conjugate gradient iteration, and the trace bounds computation. Numerical performances of these building blocks are investigated in this section. The timing was done on a Sun–SparcStation/330.

4.1. Periodic spline with equal weights. To illustrate the periodic algorithm, we generated $n_1 \times n_2$ images, $n_1 = n_2 = 32, 64, 128$, on $(0,1)^2$ by adding Gaussian noise with mean 0 and variance .04 to the function in (3.1). We applied the algorithm with $m = 2$ to the image. Some timing results are summarized in Table 4.1. Also included in Table 4.1 are the mean square errors of the estimates, defined by

$$\text{mse} = \frac{1}{n}\sum_{j=1}^{n}(\hat{f}(\mathbf{x}_j) - f(\mathbf{x}_j))^2,$$

where $\hat{f}$ is the estimate and f is the true function.

TABLE 4.1
Timing and mean square errors for periodic algorithm.

n_1, n_2	FFT (sec./pair)	Newton (sec./iter.)	mse
32	0.24	0.60/7	.00138
64	1.04	3.00/9	.00065
128	4.94	8.46/6	.00018

The contour plots of the true function and the reconstruction from the 32×32 noisy image corresponding to the first line of Table 4.1 are presented in Fig. 4.1.

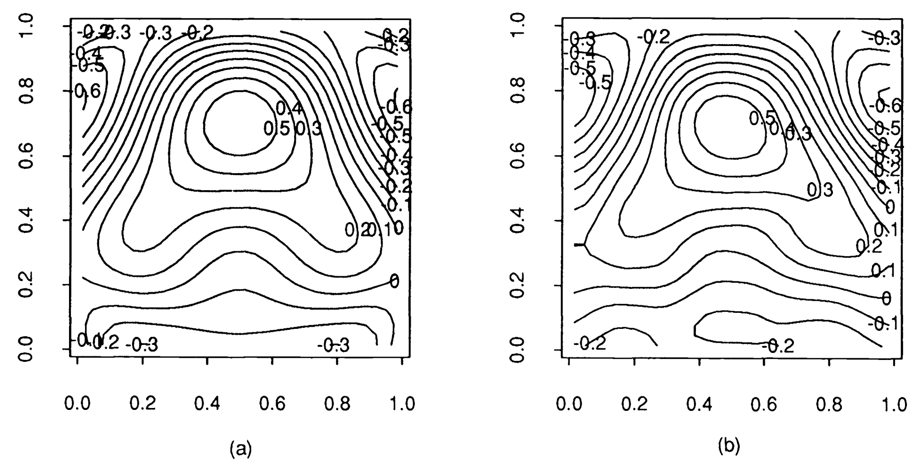


FIG. 4.1. *Reconstruction of a periodic image.* (a) *The contour of the true function.* (b) *The contour of a* 32×32 *reconstruction*

The calculation was done in double precision. The FFT was conducted using double precision modifications of routines from Swartztrauber's FFTPACK [15]. We used $n\lambda/\theta_1 = \text{tr}\Lambda_1/n_1$, $n\lambda/\theta_2 = \text{tr}\Lambda_2/n_2$, and $n\lambda/\theta_{1,2} = \text{tr}(\Lambda_1 \otimes \Lambda_2)/n_1n_2$ as starting values for the Newton iteration and the iteration converged in a few steps, where convergence meant that the achieved score V was estimated to be less than $1.001V_{\text{min}}$, where V_{min} is the true minimum. Note that the bottom of V could be very flat and one would be happy anywhere at the bottom. See [11] for a discussion. It can be seen that the timing for FFT went up at a rate of roughly $O(n \log n)$, and the timing for the Newton iteration (per step) went up at a rate of roughly $O(n)$. It can be seen that a pair of FFTs is roughly equivalent to three Newton steps. The Newton iteration usually dominates the execution, since it needs more than three steps. The practical order of the algorithm is slightly higher than $O(n)$.

4.2. General spline with equal weights. To test the general algorithm with preconditioned conjugate gradient iteration, we generated $n_1 \times n_2$ images by adding Gaussian noise from $N(0, .04)$ to Franke's [5] test function

$$.75\exp[-((9x_1-2)^2+(9x_2-2)^2)/4]+.75\exp[-(9x_1+1)^2/49-(9x_2+1)^2/10]$$
$$+.5\exp[-(9x_1-7)^2-(9x_2-3)^2/4]-.2\exp[-(9x_1-4)^2-(9x_2-7)^2/4].$$

We first applied the periodic algorithm to estimate the θ's and a starting $n\lambda$. To help the periodic algorithm choose reasonable smoothing parameters, we detrended the images at the outset by $P_S^{\perp}\mathbf{y}$. Since the GCV method chooses λ's and θ's that approximately minimize the mean square error of the smoothed image and since a periodic smoothing of a nonperiodic image generally introduces large errors on the boundary, the periodic GCV algorithm tends to choose smoothing parameters smaller than appropriate. It is known that the GCV score (and the mean square error) is less sensitive to the ratios of the smoothing parameters than to their magnitudes, so we decided to fix the θ's as returned from the periodic algorithm but to do a crude search on $n\lambda$ on the upper side of the $n\lambda$ returned from the periodic algorithm (say, $n\lambda_0$). We have run the preconditioned conjugate gradient iteration for $n\lambda = n\lambda_0, 3.18(n\lambda_0), 10(n\lambda_0), 31.8(n\lambda_0), 100(n\lambda_0)$. Table 4.2 summarizes some timing results, the GCV bounds, and the mean square errors.

TABLE 4.2
Timing, GCV bounds, and mean square errors for general algorithm.

n_1, n_2	$n\lambda$	FFT (sec./pr)	Ntn (sec./iter.)	CG (sec./iter.)	Tr (sec.)	100V	mse
32	$n\lambda_0$	0.25	0.60/7	15.84/22	0.40	$4.2837 \pm .0178$	.00151
32	$3.18(n\lambda_0)$	0.30	0.61/7	22.32/30	0.40	$4.2343 \pm .0172$	.00128
32	$10(n\lambda_0)$	0.25	0.69/7	17.41/24	0.40	$4.2547 \pm .0170$	.00149
32	$31.8(n\lambda_0)$	0.26	0.61/7	12.78/17	0.41	$4.4092 \pm .0172$	.00283
64	$(n\lambda_0)$	1.06	2.48/7	89.45/31	2.11	$4.1150 \pm .0042$	.00096
64	$3.18(n\lambda_0)$	1.07	2.42/7	125.32/45	2.21	$4.0986 \pm .0041$	.00077
64	$10(n\lambda_0)$	1.07	2.54/7	90.49/32	2.14	$4.1095 \pm .0041$	.00080
64	$31.8(n\lambda_0)$	1.09	2.81/7	82.25/28	2.42	$4.1802 \pm .0041$	.00138
128	$(n\lambda_0)$	5.16	7.32/5	1005.78/87	13.55	$4.1517 \pm .0011$	.00063
128	$3.18(n\lambda_0)$	5.17	7.30/5	865.44/76	13.58	$4.1375 \pm .0011$	.00043
128	$10(n\lambda_0)$	4.99	7.22/5	697.60/61	13.07	$4.1279 \pm .0010$	.00030
128	$31.8(n\lambda_0)$	4.85	7.19/5	813.86/72	12.21	$4.1229 \pm .0010$	.00022
128	$100(n\lambda_0)$	4.85	7.20/5	722.20/66	12.80	$4.1250 \pm .0010$	.00023

The contours of the true function and the 32×32 reconstruction corresponding to the second line of Table 4.2 are plotted in Fig. 4.2.

It can be seen that the GCV bounds provided very accurate approximation to the true score, and the GCV score followed the mean square error closely. The execution time for computing the trace bounds was negligible compared to the time for the conjugate gradient iterations, so speed-wise the bounds dominate Girard's method. To compare the precision of the two methods, we computed ten samples of Girard's approximate GCV score using $n\mathbf{c}_\epsilon^T\epsilon/\epsilon^T\epsilon$ as the trace for the case listed in the first line of Table 4.2. The mean of the 10 samples was .042707; the standard deviation was .000625. As a comparison, the 100 percent confidence interval for V using the trace bounds is $(.042659, .043015)$ with a width .000356. Only one sample of Girard's approximation out of the ten fell into this interval. Nevertheless, this comparative study by no means impairs the merits of Girard's method, which applies to more general problems and performs satisfactorily as a λ-selector on many problems when a single set of ϵ is used at all the λ values.

We finally remark on some practicalities about the algorithm. Comparing the two convenient choices, the preconditioner C of (2.8) performed much better than

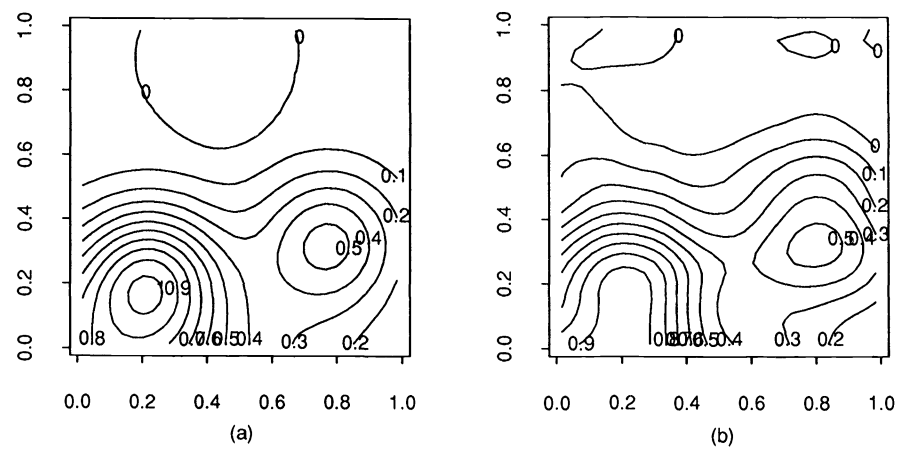


FIG. 4.2. *Reconstruction of Franke's function.* (a) *The contour of the true function.* (b) *The contour of a* 32 × 32 *reconstruction.*

the diagonal of $P_{\tilde{S}}^{\perp}\tilde{Q}P_{\tilde{S}}^{\perp} + n\lambda I$ in our test runs; we have not been able to locate other convenient preconditioners. We did have difficulty with the convergence of the conjugate gradient iteration when applying the algorithm to certain undetrended images, while detrending seemed to be effective in curing the problem. The problem was that the criterion we set (.001) seemed to be too stringent for the linear system with smoothing parameters chosen by the periodic algorithm for those undetrended images. The starting value for the conjugate gradient iteration in general does not seem to affect the speed of convergence, unless it is very close to the solution. However, double precision is essential to fast (and even eventual) convergence of the iteration.

4.3. Unequal weights. For unequal weights, we ran the algorithm on a 32 × 32 image generated by adding noise to Franke's test function. The noises were Gaussian from $N(0, .04)$ scaled by weights $w_{i,j}^{-1/2}$. For generating the weights, we generated two sets of uniform deviates from $U(0,1)$, mapped them onto $(-u/2, u/2)$, $u = 0, \ldots, 5$, and took these as $\log w_{i,j}$, where the unbalancedness increases as u increases and $u = 0$ corresponds to equal weights. This procedure generated 11 sets of weights in two groups (including the equal weights). Before applying the algorithms, we first scaled the weights to make their geometric mean equal to 1. We ran the equal weights periodic algorithm on $P_{WS}^{\perp}W^{1/2}\mathbf{y}$ to choose θ's and fix an $n\lambda_0$, then we applied the conjugate gradient iteration (without preconditioning) to solve (2.9) with $n\lambda = n\lambda_0, 3.18n\lambda_0, 10n\lambda_0, 31.8n\lambda_0$, and $100n\lambda_0$. The GCV scores based on the trace bounds using an approximate weight matrix $W_1 \otimes W_2$ were computed, where W_1 and W_2 are row-wise and column-wise geometric means of W. Girard's estimates of the GCV score were also computed using two sets of ϵ for all the cases. Some timing results for the $u = 5$ weights in the first group, together with the corresponding approximate GCV bounds, Girard's GCV estimates, the mean square errors, and the weighted mean square errors, are summarized in Table 4.3, where the weighted mean square error is defined by

TABLE 4.3
Timing for unequal weights.

$n\lambda$	Ntn (sec./iter.)	CG (sec./iter.)	Tr (sec.)	$100V$	$100V_{g,1}$	$100V_{g,2}$	mse	wmse
$n\lambda_0$	0.52/6	31.95/61	0.42	$3.7844 \pm .0016$	3.8436	4.1274	.00179	.00454
$3.18(n\lambda_0)$	0.53/6	21.97/40	0.42	$3.8998 \pm .0016$	3.9491	4.1735	.00137	.00350
$10(n\lambda_0)$	0.51/6	15.13/27	0.42	$4.0245 \pm .0016$	4.0725	4.2372	.00118	.00312
$31.8(n\lambda_0)$	0.51/6	9.87/17	0.42	$4.2898 \pm .0017$	4.3454	4.4561	.00161	.00420

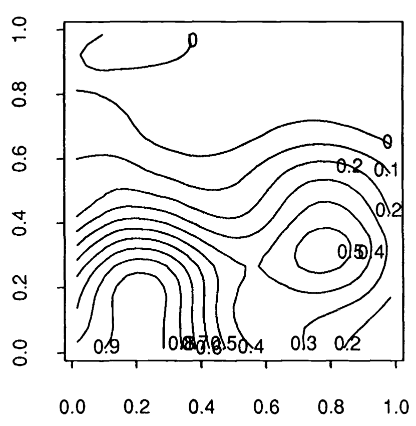


FIG. 4.3. *The contour of a* 32×32 *reconstruction of Franke's test function using unequal weights.*

$$\text{wmse} = \frac{1}{n}\sum_{j=1}^{n} w_j(\hat{f}(\mathbf{x}_j) - f(\mathbf{x}_j))^2.$$

Asymptotically, the minimizer of (2.10) is supposed to minimize the weighted mean square error; see [12] and [9].

The conjugate gradient iteration here mainly operates on real numbers and it takes less execution time per step than that of the preconditioned conjugate gradient iteration of the equal weights algorithm, which operates on complex numbers. However, the convergence is usually slower than that of the equal weights algorithm and the execution time is longer when being applied to an equal weights problem. Table 4.4 summarizes the centers of the approximate GCV bounds (V_c), the two sets of Girard's GCV estimate ($V_{g,1}$ and $V_{g,2}$), and the weighted mean square errors calculated with all the 11 sets of weights and the five relative levels of λ. It can be seen that the three GCV estimates roughly parallel each other. For the weights with large u in the first group, the GCV choices of λ tend to undersmooth the data, while the knowledge that a better λ should be on the upper side of the choice of the periodic algorithm prevents poorer choices. For the weights in the second group the GCV choices performed reasonably well at all unbalancedness levels. The contour of the reconstruction corresponding to the third line of Table 4.3 is plotted in Fig. 4.3.

5. Conclusion. In this article we have studied the computation of interaction splines when the data are sampled from a regular product mesh with noise. Algorithms

TABLE 4.4
GCV *estimates and weighted mean square errors for unequal weights.*

$n\lambda$	$100V_c$	$100V_{g,1}$	$100V_{g,2}$	1000wmse	$100V_c$	$100V_{g,1}$	$100V_{g,2}$	1000wmse
				$\log w_{i,j} \in (-0.0, 0.0)$				
$n\lambda_0$	4.284	4.232	4.289	1.504	4.284	4.232	4.289	1.504
$3.18(n\lambda_0)$	4.234	4.185	4.234	1.276	4.234	4.185	4.234	1.276
$10(n\lambda_0)$	4.255	4.214	4.252	1.495	4.255	4.214	4.252	1.495
$31.8(n\lambda_0)$	4.409	4.373	4.412	2.828	4.409	4.373	4.412	2.828
$100(n\lambda_0)$	4.655	4.619	4.670	5.074	4.655	4.619	4.670	5.074
				$\log w_{i,j} \in (-0.5, 0.5)$				
$n\lambda_0$	4.216	4.173	4.273	1.943	4.384	4.346	4.441	1.414
$3.18(n\lambda_0)$	4.190	4.143	4.228	1.585	4.335	4.292	4.377	1.190
$10(n\lambda_0)$	4.214	4.173	4.239	1.631	4.344	4.307	4.378	1.396
$31.8(n\lambda_0)$	4.362	4.327	4.380	2.727	4.481	4.448	4.514	2.689
$100(n\lambda_0)$	4.420	4.588	4.642	4.944	4.728	4.697	4.768	5.080
				$\log w_{i,j} \in (-1.0, 1.0)$				
$n\lambda_0$	4.135	4.096	4.247	2.332	4.474	4.461	4.597	1.380
$3.18(n\lambda_0)$	4.140	4.096	4.223	1.959	4.427	4.399	4.528	1.161
$10(n\lambda_0)$	4.215	4.177	4.274	2.193	4.435	4.407	4.518	1.398
$31.8(n\lambda_0)$	4.442	4.414	4.487	3.793	4.575	4.550	4.648	2.799
$100(n\lambda_0)$	4.741	4.721	4.782	6.246	4.843	4.820	4.914	5.498
				$\log w_{i,j} \in (-1.5, 1.5)$				
$n\lambda_0$	4.028	4.026	4.223	3.001	4.553	4.579	4.754	1.431
$3.18(n\lambda_0)$	4.067	4.043	4.216	2.389	4.513	4.508	4.680	1.194
$10(n\lambda_0)$	4.136	4.109	4.245	2.282	4.522	4.507	4.662	1.447
$31.8(n\lambda_0)$	4.345	4.325	4.424	3.468	4.670	4.657	4.790	2.978
$100(n\lambda_0)$	4.702	4.690	4.762	6.230	4.978	4.966	5.084	6.139
				$\log w_{i,j} \in (-2.0, 2.0)$				
$n\lambda_0$	3.895	3.945	4.172	3.871	4.624	4.706	4.912	1.591
$3.18(n\lambda_0)$	3.982	3.991	4.202	2.981	4.589	4.622	4.833	1.286
$10(n\lambda_0)$	4.074	4.063	4.238	2.637	4.599	4.607	4.805	1.499
$31.8(n\lambda_0)$	4.245	4.231	4.363	3.230	4.750	4.751	4.923	3.059
$100(n\lambda_0)$	4.567	4.559	4.652	5.388	5.111	5.112	5.260	5.388
				$\log w_{i,j} \in (-2.5, 2.5)$				
$n\lambda_0$	3.784	3.857	4.127	4.540	4.687	4.847	5.068	1.871
$3.18(n\lambda_0)$	3.900	3.923	4.174	3.496	4.659	4.749	4.981	1.445
$10(n\lambda_0)$	4.024	4.024	4.237	3.116	4.664	4.709	4.938	1.528
$31.8(n\lambda_0)$	4.290	4.288	4.456	4.202	4.783	4.806	5.013	2.776
$100(n\lambda_0)$	4.821	4.829	4.955	7.961	5.164	5.180	5.356	6.659

for situations of different complexities are proposed and their numerical performances are timed. An important aspect of the algorithms is the automatic smoothing parameter selection using the generalized cross-validation method, for which two methods were discussed and their merits in different situations explored. The ingredients of the algorithms have orders of $O(n)$ and $O(n \log n)$, where the $O(n \log n)$ of FFT is dominated by the $O(n)$ portion of the calculation. However, the overall execution time mainly depends on the convergence speed of the conjugate gradient iteration, which varies from problem to problem. Our simulations have demonstrated the effectiveness of the technique on nice continuous functions with Gaussian noise. Further study is needed to explore the effectiveness of the technique on other noise structures such as binary images and Poisson images.

REFERENCES

[1] N. ARONSZAJN, *Theory of reproducing kernels*, Trans. Amer. Math. Soc., 68 (1950), pp. 337–404.

[2] D. Barry, *Nonparametric Bayesian regression*, Ann. Statist., 14 (1986), pp. 934–953.

[3] Z. Chen, *Interaction spline models and their convergence rates*, Ann. Statist., 19 (1991), pp. 1855–1868.

[4] P. Craven and G. Wahba, *Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation*, Numer. Math., 31 (1979), pp. 377–403.

[5] R. Franke, *A critical comparison of some methods for interpolation of scattered data*, Tech. Rep. NPS-53-79-003, Naval Postgraduate School, Monterey, CA, 1979.

[6] D. Girard, *A fast "Monte-Carlo cross-validation" procedure for large least squares problems with noisy data*, Numer. Math., 56 (1989), pp. 1–23.

[7] ———, *Asymptotic optimality of the fast randomized versions of* GCV *and* C_L *in ridge regression and regularization*, Ann. Statist., 19 (1991), pp. 1950–1963.

[8] G. Golub and C. Van Loan, *Matrix Computation*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.

[9] C. Gu, *Adaptive spline smoothing in non-Gaussian regression models*, J. Amer. Statist. Assoc., 85 (1990), pp. 801–807.

[10] C. Gu, D. Bates, Z. Chen, and G. Wahba, *The computation of* GCV *functions through Householder tridiagonalization with application to the fitting of interaction spline models*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 457–480.

[11] C. Gu and G. Wahba, *Minimizing* GCV/GML *scores with multiple smoothing parameters via the Newton method*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 383–398.

[12] F. O'Sullivan, B. Yandell, and W. Raynor, *Automatic smoothing of regression functions in generalized linear models*, J. Amer. Statist. Assoc., 81 (1986), pp. 96–103.

[13] F. O'Sullivan, *Discretized Laplacian smoothing by Fourier methods*, J. Amer. Statist. Assoc., 86 (1991), pp. 634–642.

[14] G. W. Stewart, *Introduction to Matrix Computation*, Academic Press, New York, 1973.

[15] P. N. Swartztrauber, *A package of Fortran subprograms for the fast Fourier transform of periodic and other symmetric sequences*, Tech. Rep., National Center for Atmospheric Research, Boulder, CO, 1979.

[16] G. Wahba, *Partial and interaction splines for the semiparametric estimation of functions of several variables*, in Computer Science and Statistics: Proceedings of the 18th Symposium on the Interface, T. J. Boardman, ed., American Statistical Association, Washington, D.C., 1986, pp. 75–80.

[17] ———, *Spline Models for Observational Data*, CBMS–NSF Regional Conference Series in Applied Mathematics, Vol. 59, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 231–238, January 1993

A COLLECTION OF PROBLEMS FOR WHICH GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING IS UNSTABLE*

STEPHEN J. WRIGHT†

Abstract. A significant collection of two-point boundary value problems is shown to give rise to linear systems of algebraic equations on which Gaussian elimination with row partial pivoting is unstable when standard solution techniques are used.

Key words. two-point boundary value problems, Gaussian elimination, partial pivoting

AMS(MOS) subject classifications. 65F05, 65L10, 65L20

1. Introduction.

It is well known that when Gaussian elimination with row partial pivoting is applied to a $k \times k$ real matrix, a growth factor of up to 2^{k-1} may be observed in the upper-triangular factor. A matrix which achieves this upper bound, due to Wilkinson [9, p. 212], has passed into the folklore of numerical linear algebra. However, matrices that exhibit growth factors which are exponential in their dimension have apparently not previously been observed in connection with any "practical applications." In the next section, we discuss two-point boundary value problems for which standard solution techniques give rise to matrices with this undesirable property. An extreme case is derived in §3. Some random trials, reported in §4, suggest that the collection of such problems may represent a significant fraction of the set of two-point boundary value problems with coupled end conditions.

2. Examples arising from two-point boundary value problems.

Consider the general two-point boundary value problem

$$y' = M(t)y + q(t), \qquad B_a y(a) + B_b y(b) = \beta, \qquad y(t) \in \mathbf{R}^n, \tag{1}$$

and the particular problem defined by

$$n = 2, \qquad M(t) \equiv \begin{bmatrix} -\frac{1}{6} & 1 \\ 1 & -\frac{1}{6} \end{bmatrix}, \qquad a = 0,\ b = 60, \qquad B_a = B_b = I. \tag{2}$$

(The values of $q(t)$ and β are not relevant to our present discussion.) The problem defined by the data (2) is well conditioned; that is, its solution y is insensitive to perturbations in the data $M(t)$, $q(t)$, B_a, B_b, and β which define it. To show this, we can construct the analytic solution as follows: First, define $Y(t) \in \mathbf{R}^{n\times n}$ as a fundamental solution of the homogeneous ordinary differential equation $y' = M(t)y$. Defining

$$Q = B_a Y(a) + B_b Y(b)$$

and

$$G(x,t) = \begin{cases} Y(x)Q^{-1}B_a Y(a) Y^{-1}(t) & t \le x, \\ -Y(x)Q^{-1}B_b Y(b) Y^{-1}(t) & t > x, \end{cases}$$

*Received by the editors October 2, 1991; accepted for publication (in revised form) February 6, 1992. This research was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under contract W-31-109-Eng-38, and by a Raybould Fellowship from the University of Queensland, Australia.

†Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

we can write

$$y(x) = Y(x)Q^{-1}\beta + \int_a^b G(x,t)q(t)\,dt \tag{3}$$

(see, for example, [1]). Well-conditioning is usually quantified in terms of norms of the two operators in (3). For the constant-coefficient problem (2) (with $M \equiv M(t)$) we have, choosing $Y(0) = I$, that

$$\begin{aligned} Y(t) &= e^{Mt}, \qquad Q = I + e^{60M}, \\ \kappa_1 &\overset{\triangle}{=} \sup_{a\le x\le b} \|Y(x)Q^{-1}\|_\infty \approx 1, \\ \kappa_2 &\overset{\triangle}{=} (b-a)\ \sup_{a\le t,x\le b} \|G(x,t)\|_\infty \approx (b-a), \end{aligned} \tag{4}$$

and so our claim of well-conditioning is verified.

A standard algorithm for problems of the form (1) is multiple shooting [6]. In its simplest form, this algorithm proceeds by partitioning $[a,b]$ into N subintervals, that is, defining a mesh $a = t_1 < t_2 < \cdots < t_{N+1} = b$ by

$$t_i = a + ih, \qquad i = 1,\ldots,N+1, \qquad h = (b-a)/N.$$

On each subinterval $[t_i, t_{i+1}]$, the following initial value problems are solved:

$$Y_i' = M(t)Y_i, \qquad Y_i(t_i) = I, \tag{5}$$

$$y_{pi}' = M(t)y_{pi} + q(t), \qquad y_{pi}(t_i) = 0. \tag{6}$$

Defining s_i as the value of the true solution y at t_i, we note that

$$y(t) = Y_i(t)s_i + y_{pi}(t), \qquad t \in [t_i, t_{i+1}], \qquad i = 1,\ldots,N.$$

Since, clearly, $y(t)$ must be continuous across the mesh points, we have

$$Y_i(t_{i+1})s_i + y_{pi}(t_{i+1}) = s_{i+1}, \qquad i = 1,\ldots,N. \tag{7}$$

Moreover, from the boundary conditions,

$$B_a s_1 + B_b s_{N+1} = \beta. \tag{8}$$

The equations (7) and (8) yield a system of linear equations whose solution is $s_1,\ldots,$ s_{N+1}. The coefficient matrix has the general form

$$A = \begin{bmatrix} B_a & & & & B_b \\ -Y_1(t_2) & I & & & 0 \\ & -Y_2(t_3) & I & & \vdots \\ & & \ddots & \ddots & 0 \\ & & & -Y_N(t_{N+1}) & I \end{bmatrix}. \tag{9}$$

For the data (2) we have in particular that $Y_i(t_{i+1}) = e^{Mh}$, and so (9) becomes

$$\bar{A} = \begin{bmatrix} I & & & & I \\ -e^{Mh} & I & & & 0 \\ & -e^{Mh} & I & & \vdots \\ & & \ddots & \ddots & 0 \\ & & & -e^{Mh} & I \end{bmatrix}. \tag{10}$$

The conditioning of the "shooting matrix" A can be related to the conditioning of the original problem (1) by a theorem of Osborne and Mattheij, which appears as Theorem 4.11 in Lentini, Osborne, and Russell [3].

THEOREM 2.1. *Suppose that* $\|[B_a|B_b]\|_\infty \leq 1$ *and that* N *is chosen large enough that*

$$\|Y_i(t_{i+1})\|_\infty \leq K', \qquad i = 1, \ldots, N.$$

Then

$$\operatorname{cond}_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty \leq (K'+1)(\kappa_1 + \kappa_2 N/(b-a)),$$

where κ_1 *and* κ_2 *are as defined in* (4).

For (10), this bound translates to

$$\operatorname{cond}_\infty(\bar{A}) \leq (\|e^{Mh}\|_\infty + 1)(1+N)$$

for h sufficiently small. (For (2) with $N = 200$, the bound is about 459.) Suppose that h is chosen small enough that all elements of e^{Mh} are less than 1 in magnitude. This is certainly possible for (2), since

$$(11) \qquad e^{Mh} = I + Mh + O(h^2) \approx \begin{bmatrix} 1-h/6 & h \\ h & 1-h/6 \end{bmatrix}.$$

If Gaussian elimination with row partial pivoting is applied to the matrix $\bar{A}$, *no pivoting occurs*, and the following factorization is obtained:

$$(12) \quad \bar{A} = \begin{bmatrix} I & & & & \\ -e^{Mh} & I & & & \\ & -e^{Mh} & I & & \\ & & \ddots & \ddots & \\ & & & -e^{Mh} & \hat{L} \end{bmatrix} \begin{bmatrix} I & & & & I \\ & I & & & e^{Mh} \\ & & I & & e^{2Mh} \\ & & & \ddots & \vdots \\ & & & I & e^{M(60-h)} \\ & & & & \hat{U} \end{bmatrix},$$

where $\hat{L}\hat{U} = (I + e^{60M})$ with $\hat{L}$ and $\hat{U}$ lower and upper triangular, respectively. Clearly, exponential element growth has taken place in the last column of U. When $N = 200$ ($h = 0.3$), the largest element in the U factor is approximately 2.59×10^{21}.

Poor performance of Gaussian elimination is not limited to matrices derived from the multiple-shooting algorithm (5)–(8). Similar behavior also occurs when "backwards shooting" (from the right-hand endpoint of each subinterval) is used and when a midpoint-rule finite difference discretization is applied to (1), (2).

As expected, a complete pivoting strategy produces a stable factorization. For the matrix $\bar{A}$, the largest element in the computed U factor is just 1.284, and both L and U factors are sparse (density 1.41 percent for L, 1.18 percent for U), though the fill-in pattern is somewhat irregular.

The behavior exhibited in (12) will occur whenever the coefficient matrix M has negative diagonal elements, since then, provided h is sufficiently small, e^{Mh} will have all its elements less than one in magnitude and no pivoting will occur. Terms of order $e^{M(b-a)}$ will therefore appear in the U factor, and these will be large if any of the eigenvalues of M have positive real parts.

Mattheij [5] has observed that stability of the partial pivoting strategy is closely related to the following feature of the pivoting pattern: if the matrix A is regarded as a collection of $N+1$ "row blocks," each containing n rows, then *the number of rows that are pivoted between row block i and row block $i+1$ $(i=1,\ldots,N-1)$ should equal the number of eigenvalues of M whose real parts are positive.* An assumption that the pivoting pattern has this property is crucial to the analysis of Wright [11]. Although this property appears to hold for most of the standard two-point boundary test problems in the literature, the matrices M just discussed lead to shooting matrices for which it is not satisfied.

The trouble is not confined to matrices M with negative diagonal entries for which no pivoting occurs during the factorization of the shooting matrix. The coefficient matrix

$$M=\begin{bmatrix} -10 & -19 \\ 19 & 30 \end{bmatrix}$$

has two positive eigenvalues, at approximately 3.755 and 16.245. Suppose we take the remaining data as in (2), construct the shooting matrix as in (5)–(8), and apply row partial pivoting. Blowup is again observed for sufficiently small h; when the shooting matrix is factorized, only one row is pivoted between each successive pair of block rows.

Neither is the blowup behavior restricted to constant-coefficient problems (i.e., those for which $M=M(t)$ is constant on $[a,b]$). Examples similar to those in this note, but with nonconstant coefficients or nonlinear dynamics, can easily be constructed by modifying the examples above. Such examples are actually more "relevant" since, in practice, they are often solved by multiple-shooting and finite difference algorithms and therefore run the risk of exhibiting the instability just described, whereas constant-coefficient problems are usually solved by other means.

In order to produce the unstable behavior, it is necessary for the problem to have coupled end conditions. If instead the boundary conditions have the separated form

$$\bar{B}_0 x(a)=\beta_a, \qquad \bar{B}_1 x(b)=\beta_b, \qquad \bar{B}_0\in \mathbf{R}^{n_a\times n}, \bar{B}_1\in \mathbf{R}^{n_b\times n}, \qquad n_a+n_b=n,$$

the matrix (9) is a permutation of a banded matrix that has a bandwidth of $2n$. Since element growth in row partial pivoting is at worst exponential in the bandwidth, the type of growth depicted above cannot occur.

Alternative stable and efficient means for producing factorizations of the matrices (9) are available. Wright [10] has described a structured QR factorization scheme for these matrices and proved its stability. (A later modification of this scheme, based on Givens rotations rather than Householder transformations, is stable while being not much less efficient than the LU algorithm discussed above.) If speed is an important consideration, the LU factorization can be attempted in the first instance and backup strategies (such as QR factorization of A or LU factorization of A^T) can be called on only if instability is detected. LU factorization of A^T will work for the problems discussed above, though it can fail on other problems, for example, the problem

$$n=2, \qquad M(t)\equiv\begin{bmatrix} \frac{1}{6} & -1 \\ -1 & \frac{1}{6} \end{bmatrix}, \qquad a=0,\ b=60, \qquad B_a=I, \qquad B_b=2I.$$

Whether there exist problems for which LU factorization of both A and A^T is unstable is an open question.

Liu and Russell [4] have apparently observed the effects of lack of stability of LU factorization on a practical problem. They use a continuation code for parametrized ordinary differential equations (ODEs) to solve the Kuramoto–Sivashinsky equation and try various factorization techniques to perform the core operation of solving the linear equations that arise repeatedly during the computation. (The coefficient matrices of these linear systems are actually bordered versions of the shooting matrices above.) They find that the continuation algorithm is less robust when an LU algorithm with partial pivoting is used to perform the factorization than when a QR algorithm is used.

3. Fraction of maximum possible growth. When $n = 1$, it is not possible to construct an example that leads to exponential blowup. A worst case appears to be the scalar problem

$$y' = q(t), \qquad y(a) + y(b) = \beta, \qquad y(t) \in \mathsf{R},$$

which, when algorithm (5)–(8) is applied, produces a matrix for which element growth of order N occurs in the U factor.

For $n = 2$, we can investigate how closely the upper bound on element growth for matrices of size $(N+1)n$, namely, $2^{(N+1)n-1}$, is approached when the matrix has the form (9). Consider a constant-coefficient problem defined by the data

$$\text{(13)} \quad n = 2, \qquad M = \lambda \begin{bmatrix} -\alpha & 1 \\ 1 & -\alpha \end{bmatrix}, \qquad a = 0, \;\; b = L, \qquad B_a = B_b = I,$$

where $\alpha \in (0,1)$. The matrix M has eigenvalues $\lambda(-\alpha \pm 1)$. By choosing N and setting $h = L/N$, we obtain a coefficient matrix similar to $\bar{A}$ in (10). To ensure that no pivoting occurs during the factorization, we must choose N large enough that no elements of e^{Mh} exceed 1. Explicit calculation of the matrix exponential shows that this requirement is equivalent to

$$e^{-\alpha\lambda h}[e^{\lambda h} + e^{-\lambda h}] \leq 2,$$

or, if we define $X = e^{\lambda h}$,

$$p(X) = X^2 - 2X^{1+\alpha} + 1 \leq 0.$$

Since, for $\alpha \in (0,1)$, $p(0) = 1$, $p(1) = 0$, $p'(1) < 0$, and $\lim_{X \to +\infty} p(X) = +\infty$, the equation

$$p(X) = 0$$

has a solution $X(\alpha)$ that is strictly greater than 1. Since $h = L/N > 0$, this is the solution of interest to us.

If we assume that no pivoting occurs during the Gaussian elimination, it follows from (12) that the largest element in the U factor is approximately equal to the largest element in e^{ML}. A little calculation shows that this is approximately

$$E = \tfrac{1}{2} e^{\lambda L(1-\alpha)}.$$

Given the upper bound on the growth for matrices of dimension $2(L/h+1)$, namely,

$$E_{\max} = 2^{2(L/h+1)-1},$$

we have

$$\rho(\alpha) \stackrel{\triangle}{=} \frac{\log_2 E}{\log_2 E_{\max}} = \frac{\dfrac{\lambda L(1-\alpha)}{\ln 2} - 1}{2(\frac{L}{h}+1) - 1}$$

$$\approx \frac{1}{\ln 2} \frac{\lambda h(1-\alpha)}{2}$$

$$= \frac{1}{2\ln 2}(1-\alpha)\ln X(\alpha).$$

Simple analysis of $p(X)$ shows that $X(\alpha) \to +\infty$ and $(1-\alpha)\ln X(\alpha) \to \ln 2$ as $\alpha \to 1^-$. Therefore,

$$\lim_{\alpha \to 1^-} \rho(\alpha) = \frac{1}{2}.$$

In fact, we can show that $\rho(\alpha)$ is monotonic increasing on the interval $(0,1)$, with $\lim_{\alpha\to 0^+} \rho(\alpha) = 0$.

We conclude that for multiple-shooting coefficient matrices arising from (13), the observed growth factor during Gaussian elimination may be as large as approximately 2^{N+1}.

4. Discussion. In [7], Trefethen writes with reference to real $k \times k$ matrices: "Perhaps large growth rates like 2^{k-1} correspond to unstable 'modes' that are themselves somehow unstable, in the sense that computations tend to drift away from them towards stabler configurations." The "drifting away" is precisely what *fails* to happen for the matrices described above. Rather, the unstable modes are propagated and reinforced because of the presence of a recurrence in which the relationship between any two successive terms is the same. The highly specialized structure of the matrices (9) and (10) accounts for the difference between our experience with randomly generated examples (reported below) and the experience of Trefethen and Schreiber [8], who worked with randomly generated *dense* matrices and observed an average growth rate of $k^{2/3}$ for real $k \times k$ matrices when partial pivoting was used.

Higham and Higham [2] present a number of dense matrices that arise naturally in applications for which the growth factors are between $n/2$ and n when both partial and complete pivoting are used. They point out that large growth factors do not necessarily imply large backward errors in the computed solution. However, for the problems described above, we would expect both the forward and backward errors in the computed solution to be large. Consider again the example (1), (2). Because $(1+e^{60M})$ is singular in double precision arithmetic, the computed U factor has a zero element in its bottom right-hand corner. Suppose we choose $q(t)$ and β in (1) such that the true solution has $y(t) = s_i = (1,1)^T$ for all t and i, and suppose that we avoid the singularity in U by setting $\hat{s}_{N+1} = s_{N+1}$, where the hat denotes a computed quantity. If we back-substitute for the remaining $\hat{s}_i$, we obtain a relative error of 2.88×10^5 in the computed solution of (7), (8). The backward error, which is defined as

$$\frac{\|c - A\hat{x}\|_2}{\|A\|_F \|\hat{x}\|_2},$$

TABLE 1
Number of "blowups" during each of five trials, each trial consisting of 100 *randomly generated problems.*

Trial	γ	$\mu \in (10^{10}, \infty)$ / $\mu \in (10^3, 10^{10}]$ $n = 2$	$n = 4$	$n = 6$
A	1	0/0	0/0	0/0
B	10	2/2	2/4	4/2
C	100	3/3	4/5	6/1
D	.1	0/0	0/0	0/0
E	2 ($B_a = B_b = I$)	2/2	5/5	10/6

where $\hat{x}$ is the computed solution of the system $Ax = c$, is found to be 0.0286. All computations here were performed in double precision.

An interesting open question is: Among all constant-coefficient problems of the form (1), is the set of triplets (M, B_a, B_b) that give rise to matrices on which Gaussian elimination fails truly a "nontrivial" subset of

$$\{(M, B_a, B_b) \mid M, B_a, B_b \in \mathbf{R}^{n \times n}, \quad \|[B_a|B_b]\|_\infty \leq 1\}?$$

Computational experiments with randomly generated constant-coefficient problems shed a little light on this question. We generated $n \times n$ matrices M $(n = 2, 4, 6)$ by choosing each matrix element from a uniform distribution on $[-10, 10]$. In the first four sets of trials, the elements of B_a and B_b were chosen likewise and then scaled so that $\|[B_a|B_b]\|_\infty = \gamma$. We chose $\gamma = 1$, 10, 100, and 0.1 in trials A, B, C, and D, respectively. In trial E, we set $B_a = B_b = I$. In each trial, 100 randomly generated problems were solved by using (5)–(8) with 1024 subintervals, with $a = 0$ and $b = 10$. For each trial we define the following quantity μ that measures the "blowup" in the U factor

$$\mu \triangleq \max(U) \ / \ \|[B_a|B_b]\|_\infty,$$

where max (U) is the magnitude of the largest element in the upper-triangular factor. Each entry in Table 1 has two components. The first is the number of problems (out of 100) for which μ exceeded 10^{10} and the second is the number of problems for which $\mu \in (10^3, 10^{10}]$. When $\mu > 10^{10}$, the resulting loss of precision is usually large enough to destroy the accuracy of the computed solution.

Three features of Table 1 are worth noting. First, the likelihood of blowup seems to increase as n increases. This is not a surprise—as n grows we would expect a mismatch between the number of positive eigenvalues and the number of rows that are pivoted between successive blocks (see §2) to become more likely. Second, the choice $B_a = I$ (which can be obtained by a simple transformation whenever the B_a in (1) is nonsingular) seems to be particularly inappropriate, though it is the "natural" choice in many circumstances. The reason for this should be clear from the discussion near the end of §2. Third, a wise strategy appears to be to scale B_a, B_b, and β by a moderately small constant. From Theorem 2.1 and the definitions of Q and κ_1, we see that this may cause some slight deterioration in the conditioning of the matrix (9), but trials A and D indicate that exponential blowup is much less likely to occur. A "too small" choice of γ may lead to Assumption 1 in [11] being violated during factorization of the first few row blocks, but after this initial phase, the pivoting pattern exhibits the "stable" pivoting feature described in §2.

Acknowledgments. The comments of Bob Russell, Nick Higham, and anonymous referees are gratefully acknowledged.

REFERENCES

[1] 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.

[2] N. J. Higham and D. J. Higham, *Large growth factors in Gaussian elimination with pivoting*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 155–164.

[3] 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.

[4] L. Liu and R. D. Russell, *Linear system solvers for boundary value* ODEs, J. Comput. Appl. Math., to appear.

[5] R. M. M. Mattheij, *Decoupling and stability of algorithms for boundary value problems*, SIAM Rev., 27 (1985), pp. 1–44.

[6] M. R. Osborne, *On shooting methods for boundary value problems*, J. Math. Anal. Appl., 27 (1969), pp. 417–433.

[7] L. N. Trefethen, *Three mysteries of Gaussian elimination*, ACM SIGNUM Newsletter, 20 (1985), pp. 2–5.

[8] L. N. Trefethen and R. S. Schreiber, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335–360.

[9] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

[10] S. J. Wright, *Stable parallel algorithms for two-point boundary value problems*, SIAM J. Sci. Comput., 13 (1992), pp. 742–764.

[11] ———, *Stable parallel elimination for boundary value* ODEs, Tech. Rep. MCS–P229–0491, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, April 1991.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 239–247, January 1993

TIMELY COMMUNICATIONS

Under the "timely communications" policy for the SIAM Journal on Scientific 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.

AN OPTIMAL TWO-LEVEL OVERLAPPING DOMAIN DECOMPOSITION METHOD FOR ELLIPTIC PROBLEMS IN TWO AND THREE DIMENSIONS*

XIAO-CHUAN CAI†

Abstract. The solution of linear systems of algebraic equations that arise from elliptic finite element problems is considered. A two-level overlapping domain decomposition method that can be viewed as a combination of the additive and multiplicative Schwarz methods is studied. This method combines the advantages of the two methods. It converges faster than the additive Schwarz algorithm and is more parallelizable than the multiplicative Schwarz algorithm, and works for general, not necessarily self-adjoint, linear, second-order, elliptic equations. The GMRES method is used to solve the resulting preconditioned linear system of equations and it is shown that the algorithm is optimal in the sense that the rate of convergence is independent of the mesh size and the number of subregions in both R^2 and R^3. A numerical comparison with the additive and multiplicative Schwarz preconditioned GMRES is reported.

Key words. overlapping domain decomposition, elliptic equations, finite elements, iterative method

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction. The domain decomposition technique is a class of preconditioned iterative methods for solving partial differential equations and has been proved to be very effective for parallel computing. In this paper, we study a new class of methods based on the Dryja–Widlund decomposition [7], in which the usual finite element space is optimally decomposed into the sum of a finite number of uniformly overlapped, two-level subspaces. Based on this decomposition, two methods, the additive Schwarz (ASM) [3], [5], [7] and the multiplicative Schwarz methods (MSM) [2], [6], have been studied. A recent paper [4] shows that MSM, despite its having less parallelism, is substantially faster than ASM in terms of their algebraic convergence rates. In this paper, we develop a new method that can be viewed as a combination of ASM and MSM, and it converges faster than the additive Schwarz method and is more parallelizable than the multiplicative Schwarz method. If the number of processors is about the same as the number of subdomains that have the same color, which will be described in detail later, then the parallelism of the new method is as good as that of ASM. We show that the new method, accelerated by certain Krylov space-based iterative methods, such as GMRES, has an optimal convergence rate independent of the mesh sizes and the number of subproblems for general elliptic problems, not necessarily symmetric, in both two- and three-dimensional spaces. The main difference between the new method and MSM is the treatment of the coarse grid

*Received by the editors January 22, 1992; accepted for publication (in revised form) May 6, 1992.

†Department of Mathematics, University of Kentucky, Lexington, Kentucky 40506 (cai@ms.uky.edu). This research was supported in part by the National Science Foundation and the Kentucky EPSCoR Program under grant STI-9108764.

operator. There are other recently developed iterative methods that make special use of the coarse grid operator; see, e.g., [1], [11], and [12].

The paper is organized as follows. In §2, we briefly introduce the elliptic finite element problem and the Dryja–Widlund decomposition. Then, we discuss the idea of transformed systems with certain well-known examples in §3. The new method is introduced in §4, in which the convergence rate of the new method is also analyzed. In §5, we provide a numerical comparison of the new method with the additive and multiplicative Schwarz methods. We conclude the paper with a few remarks in §6.

2. Model problem and Dryja–Widlund decomposition.

Let Ω be a bounded polygonal region in R^d ($d = 2$ or 3) with boundary $\partial\Omega$. We consider the weak form of the homogeneous Dirichlet boundary value problem: Find $u \in H_0^1(\Omega)$ such that

$$b(u,v) = (f,v) \quad \forall v \in H_0^1(\Omega), \tag{1}$$

where the bilinear form $b(u,v) = a(u,v) + s(u,v)$ and

$$a(u,v) = \sum_{i,j=1}^{d} \int_\Omega a_{ij} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_j} dx \quad \text{and} \quad s(u,v) = \sum_{i=1}^{d} \int_\Omega b_i \frac{\partial u}{\partial x_i} v dx + \int_\Omega cuvdx.$$

We assume that all coefficients are sufficiently smooth, the matrix $\{a_{ij}(x)\}$ is symmetric and uniformly positive definite, and $f \in H^{-1}(\Omega)$. We also assume that the equation has a unique solution and that $b(\cdot,\cdot)$ satisfies, for some positive constants c and C,

- $c\,\|u\|_a^2 \le b(u,u)$ for all $u \in H_0^1(\Omega)$,
- $\mid b(u,v) \mid \le C\,\|u\|_a \|v\|_a$ for all $u,v \in H_0^1(\Omega)$.

Here $\|\cdot\|_a = a(\cdot,\cdot)^{1/2}$ is the energy norm of $H_0^1(\Omega)$. We solve (1) by the Galerkin conformal finite element method. For simplicity, we use piecewise linear triangular elements in R^2 and the corresponding tetrahedral elements in R^3. Following Dryja and Widlund [7], we describe a two-level triangulation of Ω and the corresponding finite element spaces. We define $\{\Omega_i, i = 1, \ldots, N\}$ to be a shape-regular finite element triangulation of Ω, where the diameter of Ω_i is of order $O(H)$. We call Ω_i a substructure and $\{\Omega_i\}$ the coarse grid or H-level triangulation of Ω. In our second step, we further divide each Ω_i into smaller simplices of diameter $O(h)$, and the union of these forms a shape-regular finite element triangulation of Ω. We call it the fine mesh or h-level triangulation of Ω. We denote by V^H and V^h the continuous, piecewise linear finite element function spaces over the H-level and h-level triangulations of Ω, respectively. The Galerkin approximation of (1) is formulated as follows: Find $u_h^* \in V^h$, such that

$$b(u_h^*, v_h) = (f, v_h) \quad \forall v_h \in V^h. \tag{2}$$

We next describe the Dryja–Widlund decomposition of V^h. To decompose Ω into overlapping subregions, we extend each Ω_i to a larger subregion Ω_i', i.e., $\Omega_i \subset \Omega_i' \subset \Omega$. The overlap is of size $O(H)$, or more precisely $\operatorname{dist}(\partial\Omega_i' \cap \Omega, \partial\Omega_i \cap \Omega) \ge \alpha H$ for all i, for a constant $\alpha > 0$. We assume that $\partial\Omega_i'$ aligns with the h-level elements and denote $\Omega_0' = \Omega$. For each $\Omega_i', i > 0$, we define $V_i^h = \{v_h \in V^h \mid v_h(x) = 0,\ x\bar{\in}\Omega_i'\} \subset V^h$. We also use the subspace $V_0^h = V^H$. It is easy to see that 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 now regroup the subregions in terms of the following coloring strategy. Associated with the decomposition $\{\Omega_i'\}$, we define an undirected graph in which nodes represent the extended subregions and the edge intersections of the extended subregions. This graph can be colored, using colors $0, \ldots, J$, such that no connected nodes have the same color. We note that Ω_0' needs its own color. It is obvious that the coloring is not unique.

3. Transformed linear system. Let $b_i(\cdot,\cdot)$ be a bilinear form, defined on the subspace V_i^h, which we will refer to as the subspace preconditioner for $b(\cdot,\cdot)$. In this paper, we only consider two cases:

(i) $b_i(\cdot,\cdot) = b(\cdot,\cdot)$ for $i = 0, 1, \ldots, N$;

(ii) $b_0(\cdot,\cdot) = b(\cdot,\cdot)$ and $b_i(\cdot,\cdot) = a(\cdot,\cdot)$ for $i = 1, \ldots, N$.

We introduce the operator $T_i : V^h \longrightarrow V_i^h$ by

$$b_i(T_i u_h, v_h) = b(u_h, v_h) \quad \forall u_h \in V^h \quad \text{and} \quad \forall v_h \in V_i^h.$$

We note that among all these operators, T_0 is the only global operator and all the others are local. We recall that $u_h^* \in V^h$ denote the exact solution of the Galerkin equation (2). It is easy to see that the vector $T_i u_h^* \in V_i^h$ can be computed, without knowing u_h^*, by using the definition of T_i and the equation (2). As an immediate consequence, if we define

$$T = \text{poly}(T_0, T_1, \ldots, T_N)$$

as a polynomial of these T_i's such that $\text{poly}(0, \ldots, 0) = 0$, then $Tu_h^* \in V^h$ can also be computed without knowing u_h^* itself. By denoting $g = Tu_h^*$, we refer to

$$Tu_h^* = g \tag{3}$$

as the *transformed system* of (2). It is not difficult to prove the following theorem.

THEOREM 3.1. *If T is invertible, then the equation* (3) *has the same solution as the Galerkin equation* (2).

We now group these maps T_i in terms of the color that the subregion was assigned. For $j = 0, 1, \ldots, J$, we denote Q_j as the sum of all T_i's that correspond to the subregions with the jth color. In fact, $Q_0 = T_0$. We remark that N (the number of subregions) may be large, while J (the number of colors) can still be small. We next look at two special examples. The first one, which is the simplest case and the degree of $\text{poly}(\cdots)$ is one, is the additive Schwarz method, in which the operator has the form

$$T_{\text{ASM}} = Q_0 + Q_1 + \cdots + Q_J.$$

The second example is the so-called multiplicative Schwarz operator

$$T_{\text{MSM}} = I - E_{J+1},$$

where I is the identity map and $E_{J+1} = (I - Q_0)(I - Q_1) \cdots (I - Q_J)$. The degree of this polynomial depends on the number of colors, and the exact form of the polynomial depends on how the subregions are colored.

It is important to note that even if the original equation (2) is not well conditioned, the transformed systems can be uniformly well conditioned and more importantly the transformed system can be so arranged that a highly parallelizable algorithm can be developed for solving it. To build such a well-conditioned and easily parallelizable transformed system is the main purpose of this paper.

4. A new transformed system and its spectral bounds. The parallelism of MSM results mainly from the fact that, for $j \neq 0$, Q_j is a sum of some local independent subproblems that can be handled in parallel. However, the global operator $Q_0 v_h = T_0 v_h$ is very special and it cannot be handled in parallel with other local subproblems. It is not the case for ASM, in which all subproblems, including T_0, can be solved in parallel.

Motivated by the above observation, we now define an operator in which the global operator T_0 is made to be additive to the rest of the local operators:

$$T_{\text{new}} = \omega T_0 + I - E_J, \tag{4}$$

where $E_J = (I - Q_1) \cdots (I - Q_J)$ and $0 < \omega \in R$ is a balancing parameter. If we define $f_{\text{new}} = T_{\text{new}} u_h^*$, then our new algorithm can be described in the following way.

ALGORITHM. Find the solution of equation (2) by solving the transformed system

$$T_{\text{new}} u_h^* = f_{\text{new}} \tag{5}$$

with an iterative method.

We show in the next theorem that the operator T_{new} is, under certain assumptions, uniformly well conditioned. In other words, its spectral bounds are independent of the mesh parameter as well as the number of subproblems. The symmetric part of T_{new} is uniformly positive definite, which guarantees the convergence of a class of Krylov space-based iterative methods, such as the GMRES method [8], [9].

THEOREM 4.1. *There exist constants $H_0 > 0$ and $\omega > 0$, independent of h and H, such that if $H \leq H_0$, then*

$$\|T_{\text{new}}\|_a \leq C$$

and

$$a(T_{\text{new}} u_h, u_h) \geq \frac{c}{(J+1)^2} \|u_h\|_a^2 \quad \forall u_h \in V^h,$$

where $C = C(H_0)$ and $c = c(H_0)$ are positive constants independent of H and h.

In order to prove the main theorem, we need to quote some known results for the well-conditionedness of T_{MSM}.

THEOREM 4.2 (Cai and Widlund [6]). *There exist constants $H_0 > 0$, $\gamma_i > 0$, $i = 1, 2$, such that if $H \leq H_0$, then*

$$\|E_J\|_a \leq \sqrt{1 - \frac{\gamma_1 H^2}{J^2}} \quad \text{and} \quad \|E_{J+1}\|_a \leq \sqrt{1 - \frac{\gamma_2}{(J+1)^2}},$$

where $\gamma_i = \gamma_i(H_0)$ are independent of H and h.

LEMMA 4.3 (Cai and Widlund [5]). *There exists a constant $H_0 > 0$, such that if $H \leq H_0$, then for any $u_h \in V^h$,*

$$\|T_0 u_h\|_a \leq C \|u_h\|_a, \qquad \|T_0 u_h - u_h\|_{L^2} \leq CH \|u_h\|_a,$$

and

$$a(T_0 u_h, u_h) \geq \|T_0 u_h\|_a^2 - cH \|u_h\|_a^2,$$

where $c = c(H_0)$ and $C = C(H_0)$ are positive constants independent of H and h.

Proof of Theorem 4.1. It is easy to see that the following identity holds:

$$T_{\text{new}} = \omega T_0 - T_0 E_J + I - E_{J+1}. \tag{6}$$

The upper bound part of this theorem can be trivially proved by using Theorem 4.2 and Lemma 4.3.

For the lower bound part, we only prove the case where $b_i(\cdot,\cdot) = b(\cdot,\cdot)$ for $i = 0,\dots,N$. The proof for the other case can be obtained in a similar way. Directly from the identity (6), we have

$$a(T_{\text{new}}u_h, u_h) = \omega a(T_0 u_u, u_h) + a(u_h, u_h) - a(E_{J+1}u_h, u_h) - a(T_0 E_J u_h, u_h). \tag{7}$$

We now estimate the right-hand side of the above equality term by term. Following Theorem 4.2, we obtain

$$a(E_{J+1}u_h, u_h) \le (1-\tilde{c})\|u_h\|_a^2, \tag{8}$$

where the constant $\tilde{c} = 1 - \sqrt{1-\gamma_2/(J+1)^2} > 0$. It is easy to verify that

$$\begin{aligned} a(T_0 E_J u_h, u_h) &= b(T_0 u_h, T_0 E_J u_h) - s(u_h, T_0 E_J u_h) \\ &= a(T_0 u_h, T_0 E_J u_h) + s(T_0 u_h - u_h, T_0 E_J u_h). \end{aligned} \tag{9}$$

By using Lemma 4.3, Theorem 4.2, and the fact that $|s(u,v)| \le C\|u\|_{L^2}\|v\|_a$ for all $u, v \in H_0^1(\Omega)$, we have

$$|s(T_0 u_h - u_h, T_0 E_J u_h)| \le CH\|u_h\|_a^2$$

and hence

$$\begin{aligned} a(T_0 E_J u_h, u_h) &\le \|T_0 u_h\|_a \|T_0 E_J u_h\|_a + CH\|u_h\|_a^2 \\ &\le C_1\|u_h\|_a\|T_0 u_h\|_a + CH\|u_h\|_a^2 \\ &\le \frac{C_1\lambda}{2}\|u_h\|_a^2 + \frac{C_1}{2\lambda}\|T_0 u_h\|_a^2 + CH\|u_h\|_a^2, \end{aligned} \tag{10}$$

where λ is an arbitrary positive constant. By taking $\lambda = \tilde{c}/C_1$, we have

$$a(T_0 E_J u_h, u_h) \le \frac{\tilde{c}}{2}\|u_h\|_a^2 + \frac{C_1^2}{2\tilde{c}}\|T_0 u_h\|_a^2 + CH\|u_h\|_a^2. \tag{11}$$

Taking all the above estimates (7), (8), (11), and the last inequality of Lemma 4.3 into account, we have

$$\begin{aligned} a(T_{\text{new}}u_h, u_h) \ge{}& \tilde{c}\|u_h\|_a^2 + \omega\|T_0 u_h\|_a^2 - CH\omega\|u_h\|_a^2 \\ & -\frac{\tilde{c}}{2}\|u_h\|_a^2 - \frac{C_1^2}{2\tilde{c}}\|T_0 u_h\|_a^2 - CH\|u_h\|_a^2. \end{aligned} \tag{12}$$

Therefore, if we choose $\omega = C_1^2/(2\tilde{c})$, then

$$a(T_{\text{new}}u_h, u_h) \ge \frac{\tilde{c}}{2}\|u_h\|_a^2 - CH\|u_h\|_a^2. \tag{13}$$

Thus, if H is small enough, we have

$$a(T_{\mathrm{new}}u_h, u_h) \geq \frac{\tilde{c}}{4}\|u_h\|_a^2 \geq \frac{\gamma_2}{8(J+1)^2}\|u_h\|_a^2, \tag{14}$$

which completes the proof of the main theorem. □

A remark is in order here about the choice of ω. ω does not depend on the size of the linear system, nor the number of subproblems. Our numerical experiments (cf. the next section) show that the algorithm is not very sensitive to ω. In fact, $\omega = 1$ has always given us better convergence than ASM.

5. Numerical experiments and comparison with ASM and MSM.

In this section, we first briefly discuss the parallel complexity of the new algorithm as compared with ASM and MSM, and then present some numerical results.

Let us make some basic assumptions before providing a parallel complexity analysis with p parallel processors. In this paper, we only focus on these computer architecture independent factors. We assume that the communication, synchronization, and load balancing costs can be ignored, and also that each subproblem is solved by using only one processor.

Furthermore, we assume that all interior problems, defined on any extended substructures, are of relatively the same size and need t_i unit time (or number of arithmetic operations) to solve. Of course, t_i depends not only on how many unknowns each subregion has, but also on the method used to solve the interior problem. This is also true for t_c in the coarse mesh problem. Table 1 shows the parallel complexity of performing the preconditioner-vector multiplication by using multiplicative, additive, and the new Schwarz-type methods.

TABLE 1
The parallel complexity of the algorithms with p processors.

Method	# of iterations	p= # of subproblems	p = (max # of subdomains with the same color + 1)
MSM	$O(1)$	$Jt_i + t_c$	$Jt_i + t_c$
ASM	$O(1)$	$\max\{t_i, t_c\}$	$\max\{Jt_i, t_c\}$
NEW	$O(1)$	$\max\{Jt_i, t_c\}$	$\max\{Jt_i, t_c\}$

TABLE 2
Iteration counts for solving the Poisson equation ($\delta = 0$) with different h, H, and overlap sizes. Here $\omega = 1.0$.

$h^{-1} =$	32	64	128	32	64	128	64	128
Overlapping size	$H = 1/4$			$H = 1/8$			$H = 1/16$	
ovlp=h	8	8	10	8	7	7	6	6
ovlp=$2h$	8	8	8	7	7	7	6	6
ovlp=$4h$	6	7	8		7	7		6

We next present some numerical results for solving this equation where

$$-\triangle u + \delta u_x + \delta u_y = f \ \text{ in } \Omega \tag{15}$$

with $u = 0$ on $\partial\Omega$ and $\Omega = [0,1] \times [0,1]$. In all cases, the exact solution $u = e^{xy}\sin(\pi x)\sin(\pi y)$, and f can thus be set accordingly.

The unit square is subdivided into two-level uniform meshes, with h and H representing the fine and coarse mesh sizes. The elliptic operator is then discretized by the usual five-point central or upwinding difference methods over both meshes. The full GMRES method, without restarting, with zero initial guess is used for all of the transformed linear systems in the usual Euclidean norm, and the stopping criterion is the reduction of the initial preconditioned residual by five orders of magnitude in the L^2 norm.

We first test a special case $\delta = 0$. Although this is a symmetric problem, we still use GMRES as the outer iterative method. The iteration counts are given in Table 2.

Our second test problem is a nonsymmetric, constant coefficient problem. We specify the constant $\delta > 0$ in Table 3. The elliptic operator is discretized by two schemes, namely, the central difference method, for relatively small δ, and the upwind difference method, for relatively large δ.

The optimal choice of ω is not unique and $\omega = 1.0$ seems among the optimal choices for the example that we tested. An example can be found in Table 4.

We finally compare the new algorithm with ASM and MSM by listing the convergence history in Table 5. It is clear that the convergence rate of the new algorithm is faster than that of ASM but slower than MSM. Some results for the same test problems obtained using other domain decomposition methods can be found in [4].

TABLE 3

Iteration counts for solving the nonsymmetric model equation with various values of δ and two discretizations. The parameter $h = 1/128$. Here $\omega = 1.0$.

	$H = 1/4$						$H = 1/8$					
	Central difference method											
$\delta =$	1	5	10	50	100	150	1	5	10	50	100	150
ovlp=h	10	12	12	16	16	14	8	9	10	16	23	25
ovlp=$2h$	9	10	10	14	12	12	7	9	9	15	20	23
ovlp=$4h$	8	9	9	11	12	12	7	8	9	13	17	20
	Upwind difference method											
$\delta =$	10	50	100	500	1000	10000	10	50	100	500	1000	10000
ovlp=h	12	13	13	11	11	11	11	14	15	16	16	17
ovlp=$2h$	10	11	11	11	11	11	10	13	14	15	15	15
ovlp=$4h$	9	10	10	10	10	10	9	11	12	12	12	12

TABLE 4

Iteration counts for different balancing parameter ω's. Here $\delta = 10$, $h = 1/128$, $H = 1/8$, ovlp $= 2h$, and central differencing is used.

ω	0.0	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0	2.5
Iteration	17	12	10	9	9	9	10	10	10	10

TABLE 5

The maximum norm of the error, defined as the difference between the computed solution and the true solution of the continuous problem, at each step of iteration. The parameters are $h = 1/128$, $H = 1/4$, overlap $= 4h$, and $\delta = 50.0$. The central differencing is used here. For the new algorithm, $\omega = 1.0$.

Iteration	MSM	NEW	ASM
1	1.126987e−01	6.025081e−01	5.994051e−01
2	3.011373e−02	3.440657e−01	5.605597e−01
3	5.950362e−03	1.932006e−01	3.647781e−01
4	1.467230e−03	7.443918e−02	3.019285e−01
5	4.354542e−04	3.493269e−02	1.113954e−01
6	2.405614e−04	1.581771e−02	9.212396e−02
7	1.969721e−04	7.474377e−03	3.602628e−02
8		3.709754e−03	1.901591e−02
9		1.234765e−03	1.255937e−02
10		3.609956e−04	7.544490e−03
11		2.050532e−04	4.329650e−03
12			2.030623e−03
13			9.756193e−04
14			6.124153e−04
15			5.179665e−04
16			2.999394e−04
17			2.096750e−04
18			2.000241e−04

6. Concluding remarks.

In this paper, we introduced a new member in the class of Schwarz-type overlapping domain decomposition methods. This class of methods has been shown to be fast, even in the case involving boundary layers; see, e.g., the recent paper of Tang [10]. The new method shares the robustness of other Schwarz methods with added parallelism.

Acknowledgments. The author is indebted to Professors M. Dryja, O. Widlund, and B. Smith for many helpful discussions and suggestions. The author would also like to thank Professors W. Gropp and D. Keyes for providing their experimental domain decomposition codes, which inspired the new algorithm.

REFERENCES

[1] J. H. BRAMBLE, Z. LEYK, AND J. E. PASCIAK, *Iterative schemes for nonsymmetric and indefinite elliptic boundary value problems*, 1991, preprint.

[2] J. H. BRAMBLE, J. E. PASCIAK, J. WANG, AND J. XU, *Convergence estimates for product iterative methods with applications to domain decomposition and multigrid*, Math. Comp., 57 (1991), pp. 1–22.

[3] X.-C. CAI, *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.

[4] X.-C. CAI, W. D. GROPP, AND D. E. KEYES, *A comparison of some domain decomposition algorithms for nonsymmetric elliptic problems,* in Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. Chan, D. Keyes, G. Meurant, J. Scroggs and R. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[5] X.-C. CAI AND O. B. WIDLUND, *Domain decomposition algorithms for indefinite elliptic problems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 243–258.

[6] ———, *Multiplicative Schwarz algorithms for some nonsymmetric and indefinite elliptic problems*, SIAM J. Numer. Anal., to appear.

[7] M. Dryja and O. B. Widlund, *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.

[8] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric system of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[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. 865–869.

[10] W.-P. Tang, *Numerical solution of a turning point problem*, in Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. Chan, D. Keyes, G. Meurant, J. Scroggs and R. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[11] J. Xu, *A new class of iterative methods for nonselfadjoint or indefinite problems*, SIAM J. Numer. Anal., 29 (1992), pp. 303–319.

[12] J. Xu and X.-C. Cai, *A preconditioned* GMRES *method for nonsymmetric or indefinite problems*, Math. Comp., 1992, to appear.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 248–252, January 1993

A NOTE ON COMPUTING EIGENVALUES OF BANDED HERMITIAN TOEPLITZ MATRICES*

WILLIAM F. TRENCH[†]

Abstract. It is pointed out that the author's $O(n^2)$ algorithm for computing individual eigenvalues of an arbitrary $n \times n$ Hermitian Toeplitz matrix T_n reduces to an $O(rn)$ algorithm if T_n is banded, with bandwidth r.

Key words. Toeplitz, Hermitian, banded, eigenvalue, eigenvector

AMS(MOS) subject classifications. 65F15, 15A18, 15A57

In a recent paper Arbenz [2] (see also [1]) presented a method for computing the eigenvalues of a Toeplitz matrix

$$T_n = (t_{i-j})_{i,j=1}^n, \tag{1}$$

where

$$t_\nu = t_{-\nu} \quad \text{and} \quad t_\nu = 0 \quad \text{if } |\nu| > r;$$

thus T_n is symmetric. If $n > r$ (which we assume henceforth), then T_n is also banded. Following Arbenz, we will say that T_n *has bandwidth* r.

Since [2] is so recent and easily accessible, there is no need to go into the details of Arbenz's algorithm here; rather, we focus on the important point that it yields the eigenvalues of T_n with a computational cost of $O(r(n + r^2))$ flops per eigenvalue. Another approach to this problem is discussed in [3] and [6].

It seems worthwhile to point out that the quite different algorithm given by the author in [9] for finding individual eigenvalues of a full Hermitian Toeplitz matrix with $O(n^2)$ flops per eigenvalue requires only $O(rn)$ flops per eigenvalue in the banded case. Here we will give the briefest description of the algorithm that suffices to make this point. For complete details, see [9]. For related results, see [10].

Theorems 1 and 2 of [9] imply the following theorem, which is the basis for the algorithm.

THEOREM 1. *Let T_n be a Hermitian Toeplitz matrix, let T_m $(1 \le m \le n)$ be its $m \times m$ principal submatrix, and define*

$$q_m(\lambda) = \frac{p_m(\lambda)}{p_{m-1}(\lambda)}, \qquad 1 \le m \le n,$$

where

$$p_0(\lambda) = 1 \quad \textit{and} \quad p_m(\lambda) = \det[T_m - \lambda I_m], \qquad 1 \le m \le n.$$

If λ is not an eigenvalue of any of the principal submatrices $T_1, \ldots, T_{n-1}$, then $q_1(\lambda), \ldots, q_n(\lambda)$ can be computed recursively as follows. Let

$$q_1(\lambda) = t_0 - \lambda, \qquad x_{11}(\lambda) = t_1/(t_0 - \lambda).$$

*Received by the editors July 22, 1991; accepted for publication (in revised form) May 27, 1992. This work was partially supported by National Science Foundation grants DMS 8907939 and DMS 9108254.

[†]Department of Mathematics, Trinity University, 715 Stadium Drive, San Antonio, Texas 78212.

Then, for $2 \le m \le n-1$,

$$q_m(\lambda) = [1 - |x_{m-1,m-1}(\lambda)|^2] q_{m-1}(\lambda),$$

$$x_{mm}(\lambda) = (q_m(\lambda))^{-1} \left[t_m - \sum_{j=1}^{m-1} t_j x_{m-j,m-1}(\lambda) \right], \tag{2}$$

and

$$x_{jm}(\lambda) = x_{j,m-1}(\lambda) - x_{mm}(\lambda)\bar{x}_{m-j,m-1}(\lambda), \qquad 1 \le j \le m-1. \tag{3}$$

Finally,

$$q_n(\lambda) = [1 - |x_{n-1,n-1}(\lambda)|^2] q_{n-1}(\lambda).$$

Moreover, if λ *is an eigenvalue of* T_n, *then*

$$Y_n(\lambda) = \begin{bmatrix} -1 \\ X_{n-1}(\lambda) \end{bmatrix}$$

is an associated eigenvector, where

$$X_{n-1}(\lambda) = \begin{bmatrix} x_{1,n-1}(\lambda) \\ x_{2,n-1}(\lambda) \\ \vdots \\ x_{n-1,n-1}(\lambda) \end{bmatrix}.$$

Let the eigenvalues of T_n be

$$\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n,$$

and suppose that we wish to compute λ_k for a given k in $\{1, 2, \ldots, n\}$. We assume that λ_k is not an eigenvalue of any of the submatrices $T_1, \ldots, T_{n-1}$. From Sturm's theorem, the number of negative values in $\{q_1(\lambda), q_2(\lambda), \ldots, q_m(\lambda)\}$ equals the number of eigenvalues of $T_m - \lambda I_m$ less than λ. Therefore, if we can guess values a and b such that $\lambda_k \in (a, b)$, then Theorem 1 and bisection can be used to find a subinterval (α, β) of (a, b) which contains λ_k but no other eigenvalue of T_n nor any eigenvalue of T_{n-1}. Since q_n is continuous on (α, β), we can then use a more elaborate iterative rootfinder to compute λ_k as a zero of q_n. In [9] and [10] we chose the Pegasus modification [5, 7] of the rule of false position, which has order of convergence approximately 1.642. If $\{\mu_j\}$ is the sequence of iterates produced by the Pegasus computation, starting with $\mu_0 = \alpha$ and $\mu_1 = \beta$, then we terminate this phase of the computation at the first integer r such that

$$|\mu_r - \mu_{r-1}| < .5(1 + \mu_r)10^{-K}, \tag{4}$$

where K is a positive integer dictated by machine precision and accuracy requirements.

For a full Hermitian Toeplitz matrix T_n the computations in Theorem 1 require approximately n^2 flops for each λ. Therefore, the computation of each eigenvalue requires $0(n^2)$ flops, where the "constant" buried in the "0" depends on the number of iterations required for the given eigenvalue. Although this number depends upon the eigenvalue itself and on the starting values (a and b), computational experience (see [9]) shows that for a given choice of K in (4), its average value over all eigenvalues of a matrix of order n is essentially independent of n. Thus we can say that the cost of the procedure is roughly $M(K)n^2$ flops per eigenvalue. (For the computations reported in [9], $M(10) \approx 11$.) However, the point of the present note is that if T_n is banded, then the computations in Theorem 1 require only $O(rn)$ flops, so the algorithm in [9] yields eigenvalues of T_n at a cost of $O(rn)$ flops per eigenvalue. The reason for this is the following theorem, which is easily obtained from Theorem 1. We omit the proof.

THEOREM 2. *In addition to the assumptions of Theorem* 1, *let T_n have bandwidth $r < n$. Then $q_1(\lambda), \dots, q_n(\lambda)$ can be computed recursively as in Theorem* 1, *except that if $m > r$, then* (2) *and* (3) *can be replaced by*

$$x_{mm}(\lambda) = -\left(q_m(\lambda)\right)^{-1} \sum_{j=1}^{r} t_j x_{m-j,m-1}(\lambda), \tag{5}$$

and

$$x_{jm}(\lambda) = x_{j,m-1}(\lambda) - x_{mm}(\lambda)\bar{x}_{m-j,m-1}(\lambda), \quad 1 \le j \le r, \quad m-r \le j \le m-1 \tag{6}$$

if $m \ge r+1$.

It is significant that the summation in (5) involves only r products rather than $m-1$ as in (2), and we compute only $2r$ (fewer if $r < m < 2r$) components of $X_{n-1}(\lambda)$ in (6), as compared to m in (3). Therefore, Theorem 2 implies that for large n the algorithm of [9] requires approximately $(3r+1)n$ flops to compute $q_0(\lambda), q_1(\lambda), \dots, q_n(\lambda)$ for a given λ if T_n has bandwidth r. Since numerical experiments indicate no significant differences between convergence properties of the algorithm for banded and full matrices, this means that the average cost of computing a single eigenvalue of a banded Hermitian Toeplitz matrix is $M(K)(3r+1)n$ flops, where $M(10) \approx 11$ if $K = 10$ in (4).

Having obtained λ_k by computations based on Theorems 1 and 2, we have as a byproduct the first $r+1$ components,

$$y_{1n}(\lambda_k) = -1, y_{2n}(\lambda_k) = x_{1,n-1}(\lambda_k), \dots, y_{r+1,n}(\lambda_k) = x_{r,n-1}(\lambda_k),$$

and the last r components,

$$y_{n-r+1,n}(\lambda_k) = x_{n-r,n-1}(\lambda_k), \dots, y_{nn} = x_{n-1,n-1}(\lambda_k),$$

of the associated eigenvector $Y_n(\lambda_k)$. However, the last r components are not independent, since if λ_k is a simple eigenvalue of T_n, then either

$$y_{n-i+1,n}(\lambda_k) = y_{in}(\lambda_k) \quad \text{or} \quad y_{n-i+1,n}(\lambda_k) = -y_{in}(\lambda_k), \quad 1 \le i \le n$$

(see [4]). In any case, even if λ_k has multiplicity greater than one, the first r components of $Y_n(\lambda_k)$ determine the rest. To see this, we recall from [8] that the components of $Y_n(\lambda_k)$ satisfy the difference equation

$$\sum_{j=-r}^{r} t_j y_{i+j,n}(\lambda_k) = \lambda_k y_{in}(\lambda_k), \qquad 1 \le i \le n, \tag{7}$$

subject to the boundary conditions

$$y_{in}(\lambda_k) = 0, \qquad -r+1 \leq i \leq 0, \tag{8}$$

and

$$y_{in}(\lambda_k) = 0, \qquad n+1 \leq i \leq n+r.$$

Therefore, if

$$y_{1n}(\lambda_k), \ldots, y_{rn}(\lambda_k) \tag{9}$$

are known, then the remaining components of $Y_n(\lambda_k)$ can in principle be obtained by treating (8) as initial conditions and computing recursively from (7):

$$y_{i+r,n}(\lambda_k) = -\frac{1}{t_r} \sum_{j=-r}^{r-1} (t_j - \delta_{0j}\lambda_k) y_{i+j,n}(\lambda_k), \qquad i \geq 1. \tag{10}$$

Since the zeros of the characteristic polynomial

$$P(z) = \sum_{j=-r}^{r} t_j z^j - \lambda_k$$

of (7) occur in reciprocal pairs, the recursion (10) is unstable and therefore computationally useless; nevertheless, it proves our assertion that the components (9) completely determine $Y_n(\lambda_k)$.

From this it seems reasonable to make the (admittedly vague) conjecture that any meaningful question depending upon the eigenvector $Y_n(\lambda_k)$ can in principle be resolved from a knowledge of the components in (9), without actually computing the remaining ones. However, if the remaining components are required, then it is useful to recall from [9] that $X_{n-1}(\lambda_k)$ is the solution of the banded Toeplitz system

$$(T_{n-1} - \lambda_k I_{n-1})X = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_r \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

This is a tractable problem, since there are several well-known fast algorithms for solving banded Toeplitz systems.

REFERENCES

[1] P. Arbenz, *Computing eivenvalues of banded symmetric Toeplitz matrices*, Z. Angew. Math. Mech., 70 (1990), pp. 595–597.

[2] ———, *Computing eigenvalues of banded symmetric matrices*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 743–754.

[3] J. L. Barlow, *Error analysis of update methods for the symmetric eigenvalue problem*, SIAM J. Matrix Anal. Appl., 14 (1993), to appear.

[4] A. Cantoni and F. Butler, *Eigenvalues and eigenvectors of symmetric centrosymmetric matrices*, Linear Algebra Appl., 13 (1976), pp. 275–288.

[5] M. DOWELL AND P. JARATT, *The "Pegasus" method for computing the root of an equation*, BIT, 12 (1972), pp. 503–508.

[6] S. L. HANDY AND J. L. BARLOW, *The numerical solution of solution of banded, Toeplitz eigenvalue problems*, Tech. Rep. CS–90-47, The Pennsylvania State University, University Park, PA, October 1990.

[7] A. RALSTON AND P. RABINOWITZ, *A First Course in Numerical Analysis*, 2nd ed., McGraw–Hill, New York, 1978.

[8] W. F. TRENCH, *On the eigenvalue problem for Toeplitz band matrices*, Linear Algebra Appl., 64 (1985), pp. 199–214.

[9] ———, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 135–146.

[10] ———, *Numerical solution of the eigenvalue problem for efficiently structured Hermitian matrices*, Linear Algebra Appl., 154–156 (1991), pp. 415–432.

SIAM J. SCI. COMPUT.
Vol. 14, No. 1, pp. 253–257, January 1993

EXPLOITING STRUCTURAL SYMMETRY IN A SPARSE PARTIAL PIVOTING CODE*

STANLEY C. EISENSTAT† AND JOSEPH W. H. LIU‡

Abstract. This short communication shows how to take advantage of structural symmetry to improve the performance of a class of partial pivoting codes for the LU factorization of large sparse unsymmetric matrices. Experimental results demonstrate the effectiveness of this technique in reducing the overall factorization time.

Key words. sparse LU factorization, partial pivoting, structural symmetry

AMS(MOS) subject classifications. 65F05, 65F50

1. Introduction.

Many implementations of sparse LU factorization with partial pivoting compute the factors one row or column at a time. Each step involves both symbolic operations (to determine the nonzero structure) and numeric operations. With the development of fast floating-point hardware and vector processors, the symbolic operations have come to represent a nontrivial fraction of the overall factorization time. Thus any sizable reduction in this symbolic overhead would have a significant impact.

The technique of *symmetric reduction* [4] exploits structural symmetry to decrease the amount of structural information required for the symbolic factorization of a sparse unsymmetric matrix (i.e., for obtaining the nonzero structures of the factor matrices). This has the practical advantage of decreasing the run-time.

In this short communication, we show how to use symmetric reduction to improve the performance of a class of partial pivoting codes for the LU factorization of large sparse unsymmetric matrices, in particular, Sherman's NSPFAC (a more recent version of NSPIV [8]) and a code of Gilbert and Peierls [7]. For some problems the speedup is more than a factor of two.

Notation. For an $n \times n$ matrix M and two sets I and J of subscripts, we let M_{IJ} denote the submatrix of M determined by the rows in I and the columns in J. As a special case, we let M_{I*} denote the submatrix of M determined by the rows in I.

We let $G(M)$ denote the associated directed graph. Here edges are directed from row to column; i.e., (r, c) is an edge in $G(M)$ if and only if m_{rc} is nonzero. We use the notation $r \xrightarrow{M} c$ to indicate the existence of an edge from r to c in $G(M)$, and $r \overset{M}{\Longrightarrow} c$ to indicate the existence of a path from r to c. We also adopt the convention that $i \overset{M}{\Longrightarrow} i$ for any i.

2. Unsymmetric symbolic factorization.

Let A be a sparse unsymmetric $n \times n$ matrix that can be decomposed (without pivoting) into $L \cdot U$, where L is lower triangular with unit diagonal and U is upper triangular. Let F denote the *filled matrix* $L + U$.

*Received by the editors February 15, 1992; accepted for publication June 23, 1992.

†Department of Computer Science and Research Center for Scientific Computation, Yale University, New Haven, Connecticut 06520. The research of this author was supported in part by U. S. Army Research Office contract DAAL03-91-G-0032.

‡Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3. The research of this author was supported in part by Natural Sciences and Engineering Research Council of Canada grant A5509, and by the Institute for Mathematics and its Applications, with funds provided by the National Science Foundation.

Assume that we have determined the nonzero structures of the first $k-1$ rows of L and U; i.e., letting $K = \{1, \ldots, k-1\}$ and $\overline{K} = \{k, \ldots, n\}$, we know the structure of

$$F_{K*} = L_{K*} + U_{K*} = \begin{pmatrix} L_{KK} & 0 \end{pmatrix} + \begin{pmatrix} U_{KK} & U_{K\overline{K}} \end{pmatrix}.$$

The following result relates the structures of the rows L_{k*} and U_{k*} to the existence of certain paths in $G(U_{K*})$.

THEOREM 2.1 (see [7]). *$k \xrightarrow{F} i$ if and only if $k \xrightarrow{A} m \overset{U_{K*}}{\Longrightarrow} i$ for some m.*

Thus, to determine the nonzero structure of $F_{k*} = L_{k*} + U_{k*}$, we can search $G(U_{K*})$ for nodes reachable from some node m for which $a_{km} \neq 0$.

3. Two sparse partial pivoting codes. We focus on two implementations of sparse LU factorization with partial pivoting: Sherman's NSPFAC (a descendant of NSPIV [8]) and Gilbert and Peierls's code [7] (referred to here as GP).

NSPFAC factors A by rows using column partial pivoting. While computing F_{k*}, it represents the structure of the current, partially formed row by an ordered, linked list of subscripts corresponding to nonzero columns. The linked list is initialized to the nonzero columns in A_{k*}. For each nonzero ℓ_{kj} (in increasing column order), the structural and numeric updates from U_{j*} to F_{k*} are applied in a single loop, one element at a time. The numeric update involves two levels of indirection.

Gilbert and Peierls [7] observed that it is not necessary to apply the row updates in increasing order—*any* order consistent with a *topological order* of $G(U_{KK})$ would suffice. They also noted that a depth-first search of $G(U_{K*})$ starting from the nonzero columns of A_{k*} gives the nonzero structure of F_{k*}, and that a topological ordering can be obtained as a byproduct, without additional work. Using this result, they show that GP runs in time proportional to the number of floating-point operations, a property not shared by other sparse partial pivoting codes.

In computing F_{k*},[1] GP first does a depth-first search to compute the structure of L_{k*} (but not U_{k*}) as above. Then, for each nonzero ℓ_{kj} (in topological order), it applies the structural updates from U_{j*} to U_{k*} and the numeric updates from U_{j*} to F_{k*} in a single loop, one element at a time.

To estimate the time NSPFAC and GP spend in nonnumeric computations, we wrote a sparse LU factorization code (called NF) that uses a predetermined pivot sequence and precomputed factor structures.[2] By using the same pivot sequence and factor structures as computed by NSPFAC or GP, we can measure how much time would be spent if the nonnumeric operations involving symbolic factorization and pivot selection were removed.

Table 2 gives the run-times[3] for ten problems from the Harwell–Boeing collection [3]. For each test matrix A, the rows of the matrix were preordered by a minimum degree ordering of AA^t, as suggested by George and Ng [5]. The results for the Sun SparcStation/1 show that the nonnumeric overhead can exceed 50 percent. For the

[1]Although GP computes the LU factorization by columns using row partial pivoting, to be consistent we describe the Gilbert–Peierls approach by rows. In the numerical experiments, GP factored A^t rather than A.

[2]NSPFAC scales rows by multiplying by the reciprocal of the pivot; GP scales columns by dividing by the pivot. To make the comparisons fair, we used two versions of NF.

[3]All programs were written in Fortran; use double-precision arithmetic; and were compiled with optimization enabled (`f77 -O` (SC1.0 Fortran V1.4) on the SparcStation/1, `xlf -O` (XL FORTRAN Compiler/6000 Version 2.2) on the RS/6000).

TABLE 1
Nonzeros in original and filled matrices.

Problem	n	$nz(A)$	$nz(F_{NSP})$	$nz(F_{GP})$
GEMAT11	4929	33185	79774	79757
JPWH991	991	6027	134741	131502
LNS3937	3937	25407	403017	403520
LNSP3937	3937	25407	383313	383340
MCFE	765	24382	68288	68288
ORANI678	2529	90158	262250	262365
ORSREG1	2205	14133	374957	374957
SAYLR4	3564	22316	624742	624742
SHERMAN3	5005	20033	409475	409475
SHERMAN5	3312	20793	242556	242556

TABLE 2
Time (in seconds) for NSPFAC/GP *and* NF *with the same pivot sequence.*

	SparcStation/1				RS/6000			
Problem	NSP	NF	GP	NF	NSP	NF	GP	NF
GEMAT11	2.61	1.44	3.23	1.59	1.75	0.48	1.98	0.50
JPWH991	29.79	17.75	32.54	18.75	19.53	5.20	18.97	5.10
LNS3937	68.58	39.76	73.79	42.86	43.27	11.88	45.02	12.38
LNSP3937	63.34	35.16	65.42	37.92	38.38	10.58	39.77	10.98
MCFE	7.18	4.16	7.89	4.63	4.83	1.28	4.65	1.32
ORANI678	32.17	15.64	29.41	16.81	21.82	4.87	17.78	5.13
ORSREG1	92.43	56.26	103.94	60.40	60.23	16.33	62.20	17.33
SAYLR4	168.39	102.90	189.65	110.18	110.07	29.80	118.82	30.78
SHERMAN3	97.31	58.60	107.58	62.70	62.88	17.10	65.18	17.32
SHERMAN5	42.50	26.01	47.98	27.88	27.90	7.73	29.25	7.92

IBM RS/6000 Model 320, which has relatively faster (with respect to the speed of its integer unit) floating-point hardware, the nonnumeric overhead can exceed 70 percent.

4. Symmetric reduction. Theorem 2.1 characterizes the nonzero structure of F_{k*} in terms of the structure of A_{k*} and paths in the graph $G(U_{K*})$. But by removing from $G(U_{K*})$ edges that are not needed to preserve the set of paths, a process called *transitive reduction* [1], we can decrease the amount of searching required to determine the structure.

If we remove all such redundant edges, then we get the *elimination dag* (*directed acyclic graph*) [6], the minimal subgraph that preserves paths. However, if we remove fewer redundant edges, we will still preserve the set of paths. The search time will be larger than for the elimination dag, but the total time (including the time for the reduction) may be less.

Symmetric reduction [4] is based on structural symmetry in the filled matrix F. The symmetric reduction of $G(U_{K*})$ is obtained by deleting all edges (i, m) for which $\ell_{ji} * u_{ij} \neq 0$ for some $j < \min\{k, m\}$. In effect, all nonzeros to the right of the first symmetric nonzero are deleted; if no such symmetric nonzero exists, then all nonzero entries are kept. We denote the resulting symmetrically reduced matrix by $\widetilde{U}_{K*}$.

Figure 1 shows the structures of two partial factor matrices $F_{K_4,*}$ and $F_{K_5,*}$, where $K_4 = \{1,2,3,4\}$ and $K_5 = \{1,2,3,4,5\}$. We use "•" to indicate a nonzero entry in the original matrix, and "∘" an entry that fills in. Since $\ell_{41} * u_{14}$ is the only symmetric nonzero pair in $F_{K_4,*}$, only the nonzeros to the right of u_{14} are pruned from $U_{K_4,*}$ to get $\widetilde{U}_{K_4*}$. On the other hand, there are two more symmetric nonzero pairs in $F_{K_5,*}$, $\ell_{52} * u_{25}$ and $\ell_{54} * u_{45}$, so that nonzeros are pruned in rows 2 and 4 to get $\widetilde{U}_{K_5,*}$.

$$F_{K_4,*} = \begin{pmatrix} 1 & & & \bullet & \bullet & & \bullet \\ & 2 & & & \bullet & \bullet & \\ \bullet & & 3 & \circ & \circ & & \circ \\ \bullet & & & 4 & \circ & \bullet & \circ \end{pmatrix} \qquad \widetilde{U}_{K_4,*} = \begin{pmatrix} 1 & & & \bullet & & & \\ & 2 & & & \bullet & \bullet & \\ & & 3 & \circ & \circ & & \circ \\ & & & 4 & \circ & \bullet & \circ \end{pmatrix}$$

$$F_{K_5,*} = \begin{pmatrix} 1 & & & \bullet & \bullet & & \bullet \\ & 2 & & & \bullet & \bullet & \\ \bullet & & 3 & \circ & \circ & & \circ \\ \bullet & & & 4 & \circ & \bullet & \circ \\ \bullet & \bullet & & \circ & 5 & \circ & \circ \end{pmatrix} \qquad \widetilde{U}_{K_5,*} = \begin{pmatrix} 1 & & & \bullet & & & \\ & 2 & & & \bullet & & \\ & & 3 & \circ & \circ & & \circ \\ & & & 4 & \circ & & \\ & & & & 5 & \circ & \circ \end{pmatrix}$$

FIG. 1. *An example to illustrate symmetric reduction.*

TABLE 3
Normalized time for the original and two modified versions of NSPFAC/GP.

	SparcStation/1						IBM RS/6000					
Problem	NSP	Red	Mod	GP	Red	Mod	NSP	Red	Mod	GP	Red	Mod
GEMAT11	1.81	1.76	1.56	2.03	1.69	1.64	3.65	2.50	2.29	3.96	2.64	2.50
JPWH991	1.68	1.27	1.09	1.74	1.09	1.09	3.76	1.58	1.19	3.72	1.24	1.21
LNS3937	1.72	1.30	1.12	1.72	1.12	1.12	3.64	1.67	1.27	3.64	1.36	1.33
LNSP3937	1.80	1.32	1.12	1.73	1.14	1.13	3.63	1.68	1.29	3.62	1.36	1.33
MCFE	1.73	1.39	1.21	1.70	1.19	1.17	3.77	1.86	1.43	3.52	1.42	1.38
ORANI678	2.06	1.79	1.60	1.75	1.25	1.21	4.48	2.86	2.51	3.47	1.62	1.49
ORSREG1	1.64	1.27	1.07	1.72	1.08	1.08	3.69	1.58	1.17	3.59	1.20	1.18
SAYLR4	1.64	1.26	1.07	1.72	1.07	1.07	3.69	1.59	1.20	3.86	1.26	1.24
SHERMAN3	1.66	1.30	1.08	1.72	1.09	1.08	3.68	1.59	1.18	3.76	1.27	1.25
SHERMAN5	1.63	1.29	1.09	1.72	1.11	1.11	3.61	1.63	1.23	3.69	1.33	1.28
Harmonic Mean	1.73	1.37	1.17	1.75	1.16	1.15	3.75	1.78	1.37	3.68	1.40	1.36

Symmetric reduction preserves the set of paths in $G(U)$ (see [4]). The argument can be adapted to show that it also preserves the set of paths in $G(U_{K*})$. The following result is an immediate corollary of this observation and Theorem 2.1.

COROLLARY 4.1. *$k \xrightarrow{F} i$ if and only if $k \xrightarrow{A} m \overset{\widetilde{U}_{K*}}{\Longrightarrow} i$ for some m.*

5. Numerical experiments. We incorporated symmetric reduction into NSPFAC and GP. In the process, we made a number of small modifications to the codes.

In NSPFAC, we split the innermost loop so that, when applying the update from U_{j*} to F_{k*}, we complete the structural update *before* performing the numeric update. Furthermore, we removed one of the two levels of indirection from the numeric update.

In GP, we removed the structural update to U_{k*} from the innermost loop and disabled the test for accidental cancellation, for otherwise symmetric reduction might not preserve paths. Furthermore, we combined the symbolic computation of L_{k*} and U_{k*} into a single depth-first search that computes the structure of F_{k*} using Corollary 4.1.

Table 3 presents the ratios of the run-times of the original and two modified versions of NSPFAC and GP to the corresponding NF using the same pivot sequence. The versions labeled "Red" include only those changes needed to incorporate symmetric reduction; the versions labeled "Mod" also include the changes that remove one level of indirection (NSPFAC) or combine the depth-first searches (GP). As in Table 2, the rows of each test matrix A were preordered by a minimum degree ordering on AA^t.

The results show a dramatic decrease in the overall factorization time. The reduction is more pronounced on the RS/6000 due to the relatively faster floating-point hardware. An even more dramatic reduction would be expected on a vector processor.

There are other ways to improve these sparse partial pivoting codes. One is to use path-symmetric or partial path-symmetric reduction, as described in [4]. Another is to switch from nodal to supernodal elimination [2], which we expect will give a substantial improvement. A code with these features is currently under development by the authors.

Acknowledgment. The authors thank John Gilbert for making available a prerelease of sparse Matlab, which was used to generate the row orderings for the test problems, and for suggesting the notation used for edges and paths.

REFERENCES

[1] A. V. Aho, M. R. Garey, and J. D. Ullman, *The transitive reduction of a directed graph*, SIAM J. Comput., 1 (1972), pp. 131–137.

[2] 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. Supercomputer Appl., 1 (1987), pp. 10–30.

[3] I. S. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[4] S. C. Eisenstat and J. W. H. Liu, *Exploiting structural symmetry in sparse unsymmetric symbolic factorization*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 202–211.

[5] J. A. George and E. Ng, *An implementation of Gaussian elimination with partial pivoting for sparse systems*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 390–409.

[6] J. R. Gilbert and J. W. H. Liu, *Elimination structures for unsymmetric sparse* LU *factors*, Tech. Report CS-90-11, Department of Computer Science, York University, North York, Ontario, Canada, 1990.

[7] J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 862–874.

[8] A. H. Sherman, *Algorithm* 533: NSPIV, *a* FORTRAN *subroutine for sparse Gaussian elimination with partial pivoting*, ACM Trans. Math. Software, 4 (1978), pp. 391–398.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 259-278, March 1993

ANALYSIS OF THE IMPLICIT EULER LOCAL UNIFORM GRID REFINEMENT METHOD*

R. A. TROMPERT† AND J. G. VERWER†

Abstract. Attention is focused on parabolic problems having solutions with sharp moving transitions in space and time. An adaptive grid method is analysed that refines the space grid locally around sharp spatial transitions, so as to avoid discretization on a very fine grid over the entire physical domain. This method is based on static-regridding and local uniform grid refinement. Static-regridding means that for evolving time the space grid is adapted at discrete times. Local uniform grid refinement means that the actual adaptation of the space grid takes place using nested locally and uniformly refined grids. The present paper concentrates on stability and error analysis while using the implicit Euler method for time integration. Maximum norm stability and convergence results are proved for a certain class of linear and nonlinear partial differential equations. The central issue is a refinement condition with a strategy that distributes spatial interpolation and discretization errors in such a way that the spatial accuracy obtained is comparable to the spatial accuracy on the finest grid if this grid would be used without any adaptation. The analysis is confirmed with a numerical illustration.

Key words. partial differential equations, numerical mathematics, time-dependent problems, adaptive grid methods, error analysis

AMS(MOS) subject classifications. primary, 65M50; secondary, 65M20

1. Introduction.

Attention is focused on parabolic problems having solutions with sharp moving transitions in space and time, such as steep fronts and disappearing layers. For such problems, a space grid held fixed throughout the entire time evolution can be computationally very inefficient. We consider an adaptive grid method that refines locally around sharp spatial transitions so as to avoid discretization on a very fine grid over the entire physical domain.

Our method is based on the techniques called static-regridding and local uniform grid refinement (LUGR), as previously proposed by Berger and Oliger [3], Gropp [6]-[8], Arney and Flaherty [2], Flaherty, Moore, and Ozturan [11], Trompert and Verwer [13], and others. Static-regridding means that for evolving time the space grid is adapted at discrete times. This should be contrasted with dynamic-regridding, where the space grid moves continuously in the space-time domain. With the term LUGR we mean that the actual adaptation of the space grid takes place using local, uniform, refined grids. LUGR should be contrasted with pointwise refinement, which leads to truly nonuniform grids. In this connection, our LUGR method bears resemblance to the fast adaptive composite grid (FAC) method [10] for elliptic equations, where the basic computational objective is to solve on an irregular grid by way of regular grids only.

The idea of the method can be briefly described as follows. Given a coarse base grid and a temporal step size, nested, local, uniform subgrids are generated. These subgrids possess nonphysical boundaries and on each of these subgrids an integration is carried out. They are generated up to a level of refinement good enough to resolve the anticipated fine scale structures. Having completed the refinement for the current base space-time grid, the process is continued to the next one while the fine grid results computed at forward time levels are kept in storage so that they can be used for step continuation.

* Received by the editors August 6, 1990; accepted for publication (in revised form) March 25, 1992.
† Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands.

An attractive feature of the static-regridding approach is the possibility of dividing the solution process into the following computational procedures: spatial discretization, temporal integration, error estimation, regridding, and interpolation. Depending on the application, these individual procedures may range from simple or straightforward to very sophisticated. This flexibility is attractive since it makes it possible to treat different types of partial differential equation (PDE) problems with almost one and the same code, assuming that the grid and the associated data structure remain unchanged. Note that the choice of data structure is important for keeping the unavoidable overhead at an acceptable level, because at each time step grids may be created or removed, while communication between grids of adjacent levels of refinement frequently takes place.

The method we analyse in this paper has many similarities with the method constructed in Trompert and Verwer [13]. In fact, the grid and data structure, the spatial differencing, and the memory use are the same. However, in the present paper we concentrate on analysis rather than on construction, while using implicit Euler instead of the explicit Runge-Kutta-Chebyschev method for time integration. The main aim of this paper is to present a detailed error analysis and to prove stability and convergence for a certain class of PDEs. The central issue in this analysis is a refinement condition and a strategy that distributes spatial discretization and interpolation errors in such a way that the spatial accuracy obtained is comparable to the spatial accuracy on the finest grid if this grid would be used without any adaptation.

Section 2 is devoted to the problem class on which we will concentrate. In § 3 we introduce the tools and the formulation for the multilevel LUGR method. In § 4 we discuss the maximum norm stability of this method. We prove an unconditional stability result which is closely related to a maximum norm stability result of implicit Euler when applied on a single space grid. Section 5 is devoted to the error analysis. In this section we investigate the total local error with its component parts. Furthermore, here we introduce the refinement strategy underlying the so-called refinement condition. This condition enables us to control the contribution of the interpolation errors in favour of discretization errors. Due to this condition, we are able to prove a convergence result as if we are working on a single fixed grid. We further elaborate on this condition in § 6, where we show how to implement it for practical use. A numerical illustration of the error analysis is given in § 7. The numerical results found here are in complete agreement with the analysis. Finally, § 8 briefly discusses our future research plans.

2. The problem class.

Following the method of lines approach [12], we consider a real abstract Cauchy problem

$$u_t = L(t, u), \quad 0 < t \leqq T, \quad u(\underset{\sim}{x}, 0) = u^0(\underset{\sim}{x}), \tag{2.1}$$

where L represents a second-order partial differential operator that differentiates the (possibly vector-valued) solution $u(\underset{\sim}{x}, t)$ to its space variable $\underset{\sim}{x}$ in a space domain Ω in $\mathbb{R}$, $\mathbb{R}^2$, or $\mathbb{R}^3$. Boundary conditions are supposed to be included in the definition of L.

With (2.1) we associate a real Cauchy problem for an explicit ordinary differential equation (ODE) system in $\mathbb{R}^d$,

$$\frac{d}{dt} U(t) = F(t, U(t)), \quad 0 < t \leqq T, \quad U(0) = U^0, \tag{2.2}$$

which is defined by a finite-difference space discretization. Thus, U and F are vectors in $\mathbb{R}^d$ representing grid functions on a space grid ω covering the interior of the space domain. Each component of U and F is vector valued if u is vector valued. The dimension d is determined by the spatial dimension, the grid spacing, and the number of PDEs in (2.1). F is determined by the type of grid, by the actual finite-difference formulas and, of course, by the precise form of L and its boundary conditions. Note that boundary values have been eliminated and worked into the ODE system. In the following, our method description and analysis are centered around this system.

Next we introduce some notations and assumptions needed for further specifying (2.1) and (2.2). The symbol $\|\cdot\|$ denotes the maximum norm on the vector space $\mathbb{R}^d$ or the induced matrix norm. Throughout our analysis we will deal only with the maximum norm. The symbol $\mu[A]$ denotes the logarithmic matrix norm of the real $d\times d$ matrix $A=(a_{ij})$ associated with $\|\cdot\|$, i.e.,

$$\mu[A]=\max_i\left(a_{ii}+\sum_{j\neq i}|a_{ij}|\right); \tag{2.3}$$

$\mu[A]$ is a useful tool in the stability analysis of nonlinear, stiff ODEs and semidiscrete PDEs [4]. In this analysis, the structure of the Jacobian matrix $F'(t,\eta)=\partial F(t,\eta)/\partial\eta$ plays a decisive role.

We are now ready to list the assumptions we make in further specifying (2.1), (2.2). These assumptions are concerned with, respectively, the class of PDEs (2.1), the smoothness of u, the choice of spatial grid and actual finite-differencing, and the stability of the semidiscrete system (2.2).

(A1) The LUGR method is applicable in any number of space dimensions. Following [13], we concentrate on the two-dimensional case, while Ω is supposed to be the unit square. With minor changes Ω is allowed to be composed of a union of rectangles with sides parallel to the coordinate axes. In fact, as we will see later, refined grids normally are of this shape. In what follows, we will generally use the notation $u(x,y,t)$, rather than $u(\underset{\sim}{x},t)$.

(A2) The solution u of (2.1) uniquely exists and is as smooth as the numerical analysis requires. Specifically, for our purpose it suffices that u is a C^2-function in t and a C^4-function in (x,y).

(A3) We will invariably use uniform space grids. Thus our base grid can be written as

$$\omega=\{(x_i,y_j): x_i=ih_x, 1\leqq i\leqq M-1 \text{ and } y_j=jh_y, 1\leqq j\leqq N-1\}, \tag{2.4}$$

where $h_x=1/M$, $h_y=1/N$, and M, N are positive integers. The spatial differencing on ω is supposed to be based on three-point formulas of second-order consistency. As a rule, we use central differencing. For boundary conditions involving first-order derivatives, the one-sided, three-point formula is used.

(A4) A constant ν exists such that $\mu[F'(t,\eta)]\leqq\nu$ for all $t\in(0,T]$, $\eta\in\mathbb{R}^d$, and all grid spacings. Like (A1) and (A2), this assumption involves a restriction on the class of PDE problems. Of course, they are made only for the sake of (model) analysis. The LUGR method remains applicable in situations where these assumptions do not hold or cannot be verified. On the other hand, for interesting classes of operators, such as the scalar, nonlinear parabolic operator

$$L(t,u)=f_1(t,x,y,u,(p_1(t,x,y)u_x)_x)+f_2(t,x,y,u,(p_2(t,x,y)u_y)_y), \tag{2.5}$$

with standard restrictions on f_i and p_i, one can prove the existence of a constant ν [4].

The inequality $\mu[F'(t,\eta)]\leqq\nu$ is to be interpreted as a stability condition, both concerning the ODE system (2.2) and its implicit Euler discretization

$$U^n=U^{n-1}+\tau F(t_n,U^n),\qquad n=1,2,\dots, \tag{2.6}$$

where $\tau = t_n - t_{n-1}$ is the step size and U^n is the approximation for $U(t_n)$. This inequality enables us to formulate the following, powerful stability result for implicit Euler. Consider the perturbed form

$$\tilde{U}^n = \tilde{U}^{n-1} + \tau F(t_n, \tilde{U}^n) + r^n, \qquad n = 1, 2, \ldots, \tag{2.7}$$

where r^n is an arbitrary local perturbation and $\tilde{U}^{n-1}$, $\tilde{U}^n$ are perturbations to U^{n-1}, U^n. Then

$$\|\tilde{U}^n - U^n\| \leqq \frac{1}{1-\tau\nu} \|\tilde{U}^{n-1} - U^{n-1} + r^n\|, \qquad n = 1, 2, \ldots, \tag{2.8}$$

for all $\tau > 0$ satisfying $\tau\nu < 1$ [4]. Since ν is independent of the grid spacing, this stability inequality is valid uniformly in h_x and h_y. For $\nu = 0$ we have contractivity for all $\tau > 0$, while for $\nu < 0$ we even have damping for all $\tau > 0$. A result closely related to (2.8) will be derived in § 4.

3. The implicit Euler local uniform grid refinement method.

3.1. Outline. Although its elaboration readily becomes complicated, the idea behind LUGR is simple. Starting from ω, finer and finer uniform subgrids are created locally in a nested manner in regions of high spatial activity. These subgrids are created by bisecting sides of next coarser grid cells. A new initial-boundary value problem is solved at each subgrid, and the integration takes place in a consecutive order, from coarse to fine. Each of these integrations spans the same time interval. Required initial values are defined by interpolation from the next coarser subgrid or taken from a subgrid from the previous time step when available. Internal boundaries are treated as Dirichlet boundaries and values are also interpolated from the next coarser subgrid. The generation of subgrids is determined by the local refinement strategy and is continued until the spatial phenomena are described well enough by the finest grid.

During each time step the following operations are performed:

1. Integrate on coarse base grid.
2. Determine new finer uniform subgrid at forward time.
3. Interpolate internal boundary values at forward time.
4. Provide new initial values at backward time.
5. Integrate on subgrid using the same steplength.
6. If the desired level of refinement is reached, go to 7, else go to 2.
7. Inject fine grid values in coinciding coarser grid points.

Thus, for each time step, the computation starts at the coarse base grid using the most accurate solution available, since fine grid solution values are always injected in coinciding coarse grid points. Moreover, all subgrids are kept in storage for step continuation.

We consider the use of uniform grids attractive because uniform grids allow an efficient use of vector-based algorithms, and finite differences on uniform grids are faster and more accurate to compute than those on nonuniform grids. In this respect, the current approach is to be contrasted with pointwise refinement leading to truly nonuniform grids. Pointwise refinement techniques also require a more involved data structure [5]. On the other hand, with the LUGR method, there are nodes that exist on more than one grid at the same time, meaning that in these nodes integration takes place more than once during one time step. Hence, the total number of nodal integrations needed will be larger than on a comparable single nonuniform grid.

In [2], [3], [7] and [11] LUGR methods are examined based on noncellular refinement and truly rectangular subgrids, which may rotate and overlap to align with an evolving fine scale structure. We avoid these difficulties. Our local subgrids do not overlap, they may be disjunct, they need not be rectangles, and the actual refinement is cellular.

3.2. The mathematical formulation. LUGR methods solve PDEs on the whole domain at the coarsest grid only and on a part of the domain at finer subgrids. Our method can be interpreted as a sequence of operations on vectors in $\mathbb{R}^d$ with varying dimension d. The dimensions are time and level dependent because the number of nodes changes per level of refinement and per time step. This constitutes a problem for the formulation of the method. To bypass this difficulty, the fine grids will be expanded so that they cover the whole domain. The dimensions are then fixed per level of refinement, which facilitates the derivation of a concise mathematical formulation. We emphasise that this grid expansion is auxiliary. In actual application, only part of the expanded higher-level grids is processed.

Suppose that for a given time interval $[0, T]$ and a given base grid, l levels are needed to describe the spatial activity of a solution sufficiently accurately when integrating over the entire time interval $[0, T]$. Introduce for $k = 1, \dots, l$ the expanded uniform grids

$$\omega_k = \{(x_i, y_j): x_i = ih_{x,k}, 1 \leqq i \leqq 2^{k-1}M - 1 \text{ and } y_j = jh_{y,k}, 1 \leqq j \leqq 2^{k-1}N - 1\}, \tag{3.1}$$

where N and M are the same integers as in (2.4), and $h_{x,k} = h_x/2^{k-1}$, $h_{y,k} = h_y/2^{k-1}$. Note that for $k = 1$ the base grid $\omega_1 = \omega$ given by (2.4) is recovered.

Let the generic notation for a grid function η defined at ω_k be η_k, and let S_k denote the space of these grid functions. We then denote the semidiscrete system considered in S_k by

$$\frac{d}{dt} U_k(t) = F_k(t, U_k(t)), \quad 0 < t \leqq T, \quad U_k(0) = U_k^0. \tag{3.2}$$

Note that due to the grid expansion, only a part of the components of the ODE system (3.2) is integrated for $k > 1$ in reality.

We are now ready to formulate the implicit Euler LUGR method. The following formula defines the time step from step point t_{n-1} to t_n for l levels of refinement:

$$U_1^n = R_{l1} U_l^{n-1} + \tau F_1(t_n, U_1^n), \tag{3.3a}$$

$$U_k^n = D_k^n[R_{lk} U_l^{n-1} + \tau F_k(t_n, U_k^n)] + (I_k - D_k^n)[P_{k-1k} U_{k-1}^n + b_k^n], \tag{3.3b}$$

for $k = 2, \dots, l$, where

$U_k^n \in S_k^n$ is the approximation to u at ω_k at $t = t_n$,
$I_k : S_k \to S_k$ is the unit matrix,
$D_k^n : S_k \to S_k$ is a diagonal matrix with entries $(D_k^n)_{ii}$ either unity or zero,
$R_{lk} : S_l \to S_k$ is the natural restriction operator from ω_l to ω_k, $R_{ll} = I_l$,
$P_{k-1k} : S_{k-1} \to S_k$ is an interpolation operator from ω_{k-1} to ω_k,
$b_k^n \in S_k$ contains time-dependent terms emanating from the boundary $\partial\Omega$.

Specifically, the nonzero entries of D_k^n $(2 \leqq k \leqq l)$ are meant to determine that part of ω_k where the actual integration takes place. This integration has the fine grid solution $D_k^n R_{lk} U_l^{n-1}$ as initial function and is defined by

$$D_k^n U_k^n = D_k^n[R_{lk} U_l^{n-1} + \tau F_k(t_n, U_k^n)], \qquad k = 2, \dots, l. \tag{3.4}$$

The definition of D_k^n is provided by the refinement strategy. For the time being, there is no need to further specify D_k^n. Note that the nesting property of the integration domains is hidden in the precise definition of the matrices D_k^n. The interpolation step is defined by

$$(3.5)\qquad (I_k - D_k^n)U_k^n = (I_k - D_k^n)[P_{k-1k}U_{k-1}^n + b_k^n], \qquad k = 2, \dots, l,$$

where the grid function b_k^n contains various time-dependent terms occurring in physical boundary conditions. We need to include b_k^n because physical boundary conditions have been worked into the semidiscrete system. For the analysis to follow, b_k^n plays no role whatsoever.

The formulation (3.3a, b) automatically comprises the interpolation of boundary values at grid interfaces. This follows directly from the observation that for nodes at grid interfaces, the associated diagonal entry of D_k^n is zero (there is no integration at grid interfaces). Further, we note that (3.3) implies an order, (3.3a) is carried out for the coarse base grid and (3.3b) for $k = 2, \dots, l$ successively. Having done this, the updating will take place, meaning that U_k^n is replaced by $R_{lk}U_l^n$ from $k = l-1$ to 1. After this we move on to the next time step. Recall that, due to the grid expansion, in (3.3a, b) the interpolation is carried out for all nodal points outside the integration domain of ω_k. This enables the stability and convergence analysis to be carried out in the spaces S_k. However, in actual application, interpolation only takes place at the local subgrids. In § 6.2 it is shown that this does not interfere with the analysis.

4. Stability analysis.

4.1. Preliminaries.

Consider, along the same lines as (2.7) for $n = 1, 2, \dots$, the perturbed scheme

$$(4.1\text{a})\qquad \tilde{U}_1^n = R_{l1}\tilde{U}_l^{n-l} + \tau F_1(t_n, \tilde{U}_1^n) + r_1^n,$$

$$(4.1\text{b})\qquad \tilde{U}_k^n = D_k^n[R_{lk}\tilde{U}_l^{n-1} + \tau F_k(t_n, \tilde{U}_k^n)] + (I_k - D_k^n)[P_{k-1k}\tilde{U}_{k-1}^n + b_k^n] + r_k^n,$$

for $k = 2, \dots, l$ with local perturbations r_k^n, and introduce the errors $e_k^n = \tilde{U}_k^n - U_k^n$, for $k = 1, \dots, l$. To shorten the formulas, we introduce the auxiliary quantities e_0^n, D_1^n, and P_{01}, where $e_0^n = 0 \in S_1$, D_1^n is the unit matrix I_1, and P_{01} is the zero matrix. Then, by subtracting (3.3a, b) from (4.1a, b), we get

$$(4.2)\quad Z_k^n e_k^n = D_k^n R_{lk} e_l^{n-1} + (I_k - D_k^n)P_{k-1k}e_{k-1}^n + r_k^n, \quad n = 1, 2, \dots, \quad k = 1, \dots, l,$$

where $Z_k^n = I_k - \tau D_k^n M_k^n$ and M_k^n is the integrated Jacobian matrix

$$(4.3)\qquad M_k^n = \int_0^1 F'(t_n, \theta\tilde{U}_k^n + (1-\theta)U_k^n)\, d\theta,$$

which results from applying the mean value theorem for vector functions.

Assuming Z_k^n to be nonsingular, we can rewrite (4.2) as

$$(4.4)\qquad e_k^n = X_k^n e_{k-1}^n + \Gamma_k^n e_l^{n-1} + \phi_k^n, \quad n = 1, 2, \dots, \quad k = 1, \dots, l,$$

with

$$(4.5)\qquad \begin{aligned} X_k^n &= (Z_k^n)^{-1}(I_k - D_k^n)P_{k-1k}, \\ \Gamma_k^n &= (Z_k^n)^{-1}D_k^n R_{lk}, \\ \phi_k^n &= (Z_k^n)^{-1} r_k^n. \end{aligned}$$

Note that $X_1^n = 0$ and that the operators X_k^n, Γ_k^n are associated, respectively, to the interpolation and restriction. We can rewrite (4.4) in the standard form

$$(4.6)\qquad e_k^n = G_k^n e_l^{n-1} + \psi_k^n, \quad n = 1, 2, \dots, \quad k = 1, \dots, l,$$

where the amplification operators G_k^n and the local perturbation terms ψ_k^n are defined by

$$(4.7)\qquad \begin{aligned} &G_1^n = \Gamma_1^n, \\ &G_k^n = X_k^n G_{k-1}^n + \Gamma_k^n, \qquad k = 2, \dots, l, \end{aligned}$$

$$(4.8)\qquad \begin{aligned} &\psi_1^n = \phi_1^n, \\ &\psi_k^n = X_k^n \psi_{k-1}^n + \phi_k^n, \qquad k = 2, \dots, l. \end{aligned}$$

The error recurrence (4.6) describes the error propagation for all refinement levels. The main interest lies in the operator G_l^n and the local perturbation ψ_l^n, since coarse grid values are always updated by fine grid values. In (4.6) this is reflected by the presence of e_l^{n-1}.

The stability of the implicit Euler method in the above is contained in the following lemma.

LEMMA 4.1. *Let ν be the logarithmic norm value defined in assumption* (A4) *of* § 2. *Then,*

$$(4.9)\qquad \begin{aligned} &\|(Z_1^n)^{-1}\| \leqq \frac{1}{1-\tau\nu} \quad \forall \tau\nu < 1, \quad k = 1, \\ &\|(Z_k^n)^{-1}\| \leqq \begin{cases} 1/(1-\tau\nu) & \forall \tau\nu < 1 \quad \text{if } \nu > 0, \\ 1 & \forall \tau > 0 \quad \text{if } \nu \leqq 0, \end{cases} \quad k = 2, \dots, l. \end{aligned}$$

Proof. The result for $k = 1$ is standard since D_1^n is the unit matrix (see [4, p. 46]). The premultiplication of M_k^n for $k > 1$ with D_k^n has the effect that either entire rows of M_k^n are put to zero, or are left unchanged. From (2.3) we can then immediately deduce that for $\nu > 0$ the bound $(1-\tau\nu)^{-1}$ still holds, whereas for $\nu \leqq 0$ the zero rows introduce the bound 1. □

Observe that the replacement of the bound $(1-\tau\nu)^{-1}$ by the bound 1 for $\nu < 0$ implies that in this case we no longer exploit the damping property of implicit Euler. For the analysis to follow, this is no restriction since here we are merely interested in proving stability and convergence results. Specifically, the stability result we will prove is not dependent on the damping in implicit Euler. To shorten derivations, we first make another assumption.

(A5) The logarithmic norm bound ν from (A4) is nonpositive. Hence we now restrict ourselves to dissipative problems. This is not essential; results obtained for $\nu \leqq 0$ can be extended to the case $\nu > 0$ by inserting $(1-\tau\nu)^{-1}$ for the bound 1 any time the stability inequality $\|(Z_k^n)^{-1}\| \leqq 1$ is used.

4.2. Stability and linear interpolation. In this section we will prove a general stability result for the multilevel adaptive grid method (3.3) that is similar to the stability result (2.8) for the implicit Euler method applied without adaption.

THEOREM 4.2. *Let $\nu \leqq 0$ according to* (A5), *and suppose that linear interpolation is used. Then, for all $\tau > 0$ and all $n \geqq 1$,*

$$(4.10)\qquad \|G_k^n\| \leqq 1, \qquad k = 1, \dots, l,$$

$$(4.11)\qquad \|\psi_k^n\| \leqq \sum_{j=1}^{k} \|r_j^n\|, \qquad k = 1, \dots, l,$$

$$(4.12)\qquad \|e_l^n\| \leqq \|e_l^{n-1}\| + \sum_{k=1}^{l} \|r_k^n\|.$$

Proof. Inequality (4.12) is a trivial consequence of (4.10) and (4.11). Let us first prove (4.10). This is done by induction with respect to k. Suppose $\|G_{k-1}^n\| \leqq 1$. From (4.7) it follows that

$$\|G_k^n\| = \|X_k^n G_{k-1}^n + \Gamma_k^n\| = \|(Z_{k-1}^n)^{-1} Q_k^n\| \leqq \|Q_k^n\|, \tag{4.13}$$

where $Q_k^n = (I_k - D_k^n) P_{k-1k} G_{k-1}^n + D_k^n R_{lk}$.

Consider the ith row of this operator. Suppose $(D_k^n)_{ii} = 1$. Then

$$\sum_j |(Q_k^n)_{ij}| = \sum_j |(R_{lk})_{ij}| = 1, \tag{4.14}$$

by definition of the restriction operator R_{lk}. Next suppose $(D_k^n)_{ii} = 0$. Then

$$\sum_j |(Q_k^n)_{ij}| = \sum_j |(P_{k-1k} G_{k-1}^n)_{ij}| \leqq \|P_{k-1k} G_{k-1}^n\| \leqq \|P_{k-1k}\| \, \|G_{k-1}^n\| \leqq 1, \tag{4.15}$$

by virtue of the induction hypothesis and the norm

$$\|P_{k-1k}\| = 1 \tag{4.16}$$

of the linear interpolation operator P_{k-1k}. Combining (4.14) and (4.15) gives $\|Q_{k+1}^n\|$, and inequality (4.10) now follows from (4.13). The induction proof is finished if we can prove that $\|G_1^n\| \leqq 1$. This follows immediately from the observation that $G_1^n = \Gamma_1^n = (Z_1^n)^{-1} R_{l1}$.

There remains to prove (4.11). We have $\|\phi_k^n\| \leqq \|r_k^n\|$. It then follows from (4.8) that

$$\|\psi_k^n\| \leqq \|X_k^n\| \, \|\psi_{k-1}^n\| + \|r_k^n\|, \tag{4.17}$$

so that we are finished if we can prove that $\|X_k^n\| \leqq 1$. This is trivial due to (4.16) and $\|I_k - D_k^n\| = 1$. □

The inequality (4.12) is the counterpart of the inequality (2.8). We may conclude from Theorem 4.2 that when implicit Euler is stable and we interpolate linearly, our multilevel adaptive grid method (3.3) retains stability of implicit Euler through the bound $\|G_l^n\| \leqq 1$.

4.3. Stability and higher-order interpolation. A drawback of linear interpolation is its limited accuracy. In a genuine application, it might well be preferable to use higher-order interpolants (in [13] we successfully used fourth-order Lagrangian interpolation). Unfortunately, in this case we must have $\|P_{k-1k}\| > 1$, so that we are not able to prove the results of Theorem 4.2 when following the above method of proof. If $\|P_{k-1k}\| > 1$, then it is possible to prove (a constrained form of) stability by introducing an additional condition that underlies the intention of interpolating exclusively in low error regions. Unfortunately, this condition turns out to be of no direct practical use and is omitted here. On the other hand, numerical evidence suggests very strongly that those higher-order interpolants do not cause genuine stability problems in real application. We believe we owe this to the fact that the method interpolates in low error regions, so that, loosely speaking, this condition is satisfied implicitly.

5. Error analysis. We will present a detailed examination of the local error. From this we deduce the refinement condition which henceforth underlies the refinement strategy. This condition enables us to control the contribution of spatial interpolation errors in favour of spatial discretization errors. Due to this condition, we can prove a convergence result as if we are working on a single fixed grid. Specifically, it will be shown that the usual convergence behaviour applies and that the accuracy obtained is comparable to the accuracy obtained on the finest grid if this grid would be used without any adaptation.

5.1. The local level error. Let $u_k(t) \in S_k$ denote the pointwise restriction of the true solution $u(x, y, t)$ to ω_k. Consider (4.1). By replacing all $\tilde{U}$-values by associated u_k-values, the local perturbation r_k^n becomes the local level error at grid level k. For convenience, we will denote this error also by r_k^n:

$$r_k^n = u_k^n - D_k^n[R_{lk}u_l^{n-1} + \tau F_k(t_n, u_k^n)] - (I_k - D_k^n)[P_{k-1k}u_{k-1}^n + b_k^n], \tag{5.1}$$
$$n = 1, 2, \ldots, \qquad k = 1, \ldots, l,$$

where $u_k^n = u_k(t_n)$ and P_{01}, u_0^n, b_1^n are auxiliary and put to zero; r_k^n contains the following local error components, the local spatial error induced by the finite-difference approximation, the local temporal error of the implicit Euler method, and the interpolation error. We first discuss these different components. They are defined in the standard way by

$$\alpha_k(t) = \frac{d}{dt} u_k(t) - F_k(t, u_k(t)) \quad \text{(spatial discretization error)}, \tag{5.2}$$

$$\beta_k(t) = u_k(t) - u_k(t-\tau) - \tau \frac{d}{dt} u_k(t) \quad \text{(temporal error)}, \tag{5.3}$$

$$\gamma_k(t) = u_k(t) - P_{k-1k}u_{k-1}(t) - b_k(t) \quad \text{(interpolation error)}. \tag{5.4}$$

The grid function $b_k(t)$ in (5.4) has the same meaning as b_k^n in (3.5). In the following, we assume without loss of generality that $h_{x,k} = h_{y,k} = h_k$. In view of assumptions (A2) and (A3) in § 2, we have

$$\alpha_k(t) = \mathcal{O}(h_k^2), \qquad h_k \to 0, \tag{5.5}$$

with order constants determined by higher-order spatial derivatives of u and by PDE operator quantities. Likewise, (A2) implies $\beta_k(t) = \tau^2 C_k$ where $C_k = -\frac{1}{2}d^2u_k/dt^2$ evaluated at a time $t + (\kappa - 1)\tau$, $0 \leqq \kappa \leqq 1$. If u is a C^3-function in t, then

$$\beta_k(t) = -\frac{1}{2}\tau^2 \frac{d^2}{dt^2} u_k(t) + \mathcal{O}(\tau^3), \qquad \tau \to 0. \tag{5.6}$$

Let q denote the accuracy order of the (Lagrangian) interpolation. Then

$$\gamma_k(t) = \mathcal{O}(h_k^q), \qquad k = 2, \ldots, l, \tag{5.7}$$

and here the order constants again depend exclusively on higher spatial derivatives of u, assuming sufficient differentiability. If linear interpolation is used, then assumption (A2) implies $q = 2$ and second-order spatial derivatives determine the constants.

Now, using the relation $u_k^{n-1} = R_{lk}u_l^{n-1}$ for $k = 1, \ldots, l$, we can derive

$$r_k^n = D_k^n(\tau\alpha_k^n + \beta_k^n) + (I_k - D_k^n)\gamma_k^n, \quad n = 1, 2, \ldots, \quad k = 1, \ldots, l. \tag{5.8}$$

Note, by definition of D_k^n, that $D_k^n(\tau\alpha_k^n + \beta_k^n)$ is the restriction of the usual local discretization error $\tau\alpha_k^n + \beta_k^n$ to the integration domain of the grid ω_k, while $(I_k - D_k^n)\gamma_k^n$ represents the restriction of the interpolation error γ_k^n to the complement of this domain.

5.2. A crude global error bound. Denote the global discretization error by $e_k^n = u_k^n - U_k^n$ and suppose $e_k^0 = 0$. For any choice of D_k^n the consistency results (5.5)-(5.8) imply

$$r_k^n = \mathcal{O}(\tau h_k^2) + \mathcal{O}(\tau^2) + \mathcal{O}(h_k^q). \tag{5.9}$$

If we now suppose linear interpolation and assumption (A5), then application of (4.12) yields

$$\|e_l^n\| \leqq \|e_l^{n-1}\| + \|S^n\| \leqq \|S^1\| + \cdots + \|S^n\|, \qquad n = 1, 2, \ldots, \tag{5.10}$$

where $\|S^n\| = \mathcal{O}(\tau h_1^2) + \mathcal{O}(\tau^2) + \mathcal{O}(h_1^2)$. Here the coarsest mesh width occurs due to simply adding all normed local level errors in (4.15), including $\|r_1^n\|$. Following standard practice, we thus obtain at any fixed time point $t_n = n\tau$ the global error bound

$$\|e_l^n\| \leqq C_1\tau + C_2 h^2 + C_3 \frac{h^2}{\tau}, \tag{5.11}$$

where $h = h_1$ and C_1, C_2, and C_3 are positive constants independent of step size and mesh sizes.

The first two terms are due to the temporal integration and spatial discretization. They will vanish if mesh sizes and step size tend to zero independently of each other, thus reflecting the unconditional convergence of the method when applied without adaptation. On the other hand, if no relation is imposed between τ and h, then the third term can grow unboundedly as τ, $h \to 0$. This term is due to the interpolation. Hence, even though we have stability and consistency, this result shows that unconditional convergence cannot be hoped for. Fortunately, this conclusion is not as bad as it looks. By not specifying the matrices D_k^n and, subsequently, by adding norms of the local level errors, we have simply supposed arbitrary integration domains at all levels of refinement. This must lead to a crude error bound like (5.11). In application, the computations should be organized in such a way that the interpolation only takes place in low error regions so that the interpolation error is virtually absent. This poses the task of setting up a precise error analysis and the design of a local refinement strategy aimed at a suitable selection of the matrices D_k^n.

5.3. Local and global errors. According to (4.6), the global error e_k^n satisfies the recurrence relation

$$e_k^n = G_k^n e_l^{n-1} + \psi_k^n, \quad n = 1, 2, \ldots, \quad k = 1, \ldots, l, \tag{5.12}$$

where ψ_k^n is the local error defined by recursion (cf. (4.8), (4.5))

$$\begin{aligned} &\psi_1^n = (Z_1^n)^{-1} r_1^n, \\ &\psi_k^n = X_k^n \psi_{k-1}^n + (Z_k^n)^{-1} r_k^n, \qquad k = 2, \ldots, l. \end{aligned} \tag{5.13}$$

The operators G_k^n, X_k^n, and Z_k^n are supposed to be redefined (replace all $\tilde{U}$-values by associated u_k-values). Note that ψ_k^n is essentially different from the local level error r_k^n. While r_k^n is associated with the single kth level, ψ_k^n is associated with all levels up to this kth level according to (5.13). This recursion governs the propagation of each local level error when introducing higher and higher levels. Elaborating, it gives, for $k = 1, \ldots, l$,

$$\psi_k^n = \sum_{j=1}^{k} \left(\prod_{i=k}^{j+1} X_i^n \right) (Z_j^n)^{-1} r_j^n. \tag{5.14}$$

Next we split ψ_k^n into its temporal and spatial part denoted by, respectively, $\psi_{k,t}^n$ and $\psi_{k,s}^n$:

$$\psi_k^n = \psi_{k,t}^n + \psi_{k,s}^n, \qquad k = 1, \ldots, l, \tag{5.15}$$

and it follows from (5.8) that $\psi_{k,t}^n$ and $\psi_{k,s}^n$ are given, respectively, by

$$\psi_{k,t}^n = \sum_{j=1}^{k} \left(\prod_{i=k}^{j+1} X_i^n \right) (Z_j^n)^{-1} D_j^n \beta_j^n, \tag{5.16}$$

$$\psi_{k,s}^n = \sum_{j=1}^{k} \left(\prod_{i=k}^{j+1} X_i^n \right) (Z_j^n)^{-1} [\tau D_j^n \alpha_j^n + (I_j - D_j^n) \gamma_j^n]. \tag{5.17}$$

Let us first examine $\psi_{k,t}^n$. Since β_k^n does not depend on mesh sizes, we have $\beta_k^n = R_{lk}\beta_l^n$. Substitution into (5.16) then yields

$$\psi_{k,t}^n = \sum_{j=1}^{k} \left(\prod_{i=k}^{j+1} X_i^n \right) (Z_j^n)^{-1} D_j^n R_{lj} \beta_l^n, \tag{5.18}$$

and we see that this operator is just the amplification operator G_k^n featured in (5.12); see the recursion (4.7). In conclusion, $\psi_{k,t}^n$ satisfies

$$\psi_{k,t}^n = G_k^n \beta_l^n, \qquad k=1,\dots,l. \tag{5.19}$$

We next examine $\psi_{k,s}^n$. Using the definition of X_k^n given in (4.5), we rewrite (5.17) as

$$\begin{aligned} \psi_{k,s}^n &= (Z_k^n)^{-1}(I_k - D_k^n)P_{k-1k} \sum_{j=1}^{k-1} \left(\prod_{i=k-1}^{j+1} X_i^n \right) (Z_j^n)^{-1}[\tau D_j^n \alpha_j^n + (I_j - D_j^n)\gamma_j^n] \\ &\quad + (Z_k^n)^{-1}[\tau D_k^n \alpha_k^n + (I_k - D_k^n)\gamma_k^n] \\ &= (Z_k^n)^{-1}[\tau D_k^n \alpha_k^n + (I_k - D_k^n)\rho_k^n], \qquad k=1,\dots,l, \end{aligned} \tag{5.20}$$

where

$$\rho_k^n = \gamma_k^n + P_{k-1k}\psi_{k-1,s}^n, \qquad k=2,\dots,l, \tag{5.21}$$

and $\rho_1^n = 0$. In (5.20) the spatial local discretization error $D_k^n \alpha_k^n$ committed on the integration domain of grid ω_k is separated from the spatial local error part $(I_k - D_k^n)\rho_k^n$ defined outside this domain. Hence, ρ_k^n collects all spatial error contributions defined on the grids ω_j $(1 \leqq j \leqq k-1)$, including discretization error α_j^n and interpolation error γ_j^n, together with γ_k^n on ω_k. This separation enables us to formulate a refinement condition which ensures that when a new grid level is introduced, the spatial local accuracy outside its integration domain will be smaller than or equal to the spatial accuracy on the integration domain itself. This distribution of local space errors is desirable, as we never return to grid points lying outside a current integration domain.

The refinement condition constrains the matrices D_k^n, and is taken to be

$$\|(Z_k^n)^{-1}\tau D_k^n \alpha_k^n\| \geqq \frac{1}{c} \|(Z_k^n)^{-1}(I_k - D_k^n)\rho_k^n\|, \quad n=1,2,\dots, \quad k=2,\dots,l, \tag{5.22}$$

where $c>0$ is a constant specified in § 6. If (5.22) is true, then all errors $\psi_{k,s}^n$ satisfy

$$\|\psi_{k,s}^n\| \leqq (1+c)\|(Z_k^n)^{-1}\tau D_k^n \alpha_k^n\|, \tag{5.23}$$

and combining (5.12) with (5.15), (5.19) enables us to present the global error inequality

$$\begin{aligned} \|e_k^n\| \leqq \|G_k^n\|\,\|e_l^{n-1}\| + \|G_k^n \beta_l^n\| + (1+c)\|(Z_k^n)^{-1}\tau D_k^n \alpha_k^n\|, \\ n=1,2,\dots, \qquad k=1,\dots,l. \end{aligned} \tag{5.24}$$

The importance of the refinement condition (5.22) is reflected by the fact that in (5.24) the interpolation error contribution has been removed. This is in agreement with our goal of developing a local refinement strategy that generates refined subgrids such that the accuracy obtained on the final finest grid is comparable to the accuracy obtained if this finest grid would be used without adaptation. We will elaborate on condition (5.22) in § 6. Note that it suffices to consider (5.22) only for $k=l$, since it suffices to consider (5.23) and (5.24) for $k=l$.

5.4. Convergence and linear interpolation. Assuming linear interpolation and assumption (A5), as in § 5.2, (5.24) can be rewritten as

$$\|e_k^n\| \leqq \|e_l^{n-1}\| + \|\beta_k^n\| + (1+c)\tau\|\alpha_k^n\|, \quad n=1,2,\dots, \quad k=1,\dots,l. \tag{5.25}$$

Hence, following the same derivation as carried out for (5.11), for the highest level l the global error bound

$$\|e_l^n\| \leqq C_1\tau + C_2(1+c)h_l^2 \tag{5.26}$$

results where C_1 and C_2 are positive constants independent of step size and mesh sizes. This bound is unconditional in the sense that it assumes no relation between step size and mesh sizes and, according to our goal, the smallest mesh width h_l occurs. We have recovered an error bound similar to the standard error bound for implicit Euler when applied on a single grid.

5.5. Convergence and higher-order interpolation. As pointed out in § 4.3, for the case of higher-order Lagrangian interpolants, a powerful stability result like that of Theorem 4.2 is not available. However, assuming that higher-order interpolation in low error regions does not severely damage stability, as is strongly supported by our practical experience, it is natural to impose the refinement condition (5.22) also in the case of higher-order interpolation. Note that in the derivation of (5.22) no a priori choice was made for the interpolants.

6. The refinement condition.

6.1. Determining the integration domains. Condition (5.22) first needs to be elaborated into a workable form before it can be implemented for determining the integration domains. To begin with, we rewrite the error ρ_k^n as

$$\begin{aligned}\rho_k^n = \gamma_k^n + P_{k-1k}\sum_{j=1}^{k-1}\left(\prod_{i=k-1}^{j+1} X_i^n\right)(Z_j^n)^{-1}\tau D_j^n \alpha_j^n \\ + P_{k-1k}\sum_{j=2}^{k-1}\left(\prod_{i=k-1}^{j+1} X_i^n\right)(Z_j^n)^{-1}(I_j - D_j^n)\gamma_j^n, \qquad 2 \leqq k \leqq l.\end{aligned} \tag{6.1}$$

Next, we rewrite the first sum as

$$\begin{aligned}P_{k-1k}\sum_{j=1}^{k-1}\left(\prod_{i=k-1}^{j+1} X_i^n\right)(Z_j^n)^{-1}\tau D_j^n \alpha_j^n = P_{k-1k}(Z_{k-1}^n)^{-1}\tau D_{k-1}^n \alpha_{k-1}^n \\ + P_{k-1k}\sum_{j=2}^{k-1}\left(\prod_{i=k-1}^{j+1} X_i^n\right)(Z_j^n)^{-1}(I_j - D_j^n) \\ \cdot P_{j-1j}(Z_{j-1}^n)^{-1}\tau D_{j-1}^n \alpha_{j-1}^n,\end{aligned} \tag{6.2}$$

and substitute this expression into (6.1). It then follows that ρ_k^n can be written as

$$\rho_k^n = \lambda_k^n + P_{k-1k}\sum_{j=2}^{k-1}\left(\prod_{i=k-1}^{j+1} X_i^n\right)(Z_j^n)^{-1}(I_j - D_j^n)\lambda_j^n, \qquad k = 2, \dots, l, \tag{6.3}$$

where

$$\lambda_j^n = \gamma_j^n + P_{j-1j}(Z_{j-1}^n)^{-1}\tau D_{j-1}^n \alpha_{j-1}^n, \qquad j = 2, \dots, l. \tag{6.4}$$

The error function λ_j^n contains the interpolation error at level j and the prolongation of the spatial discretization error of level $j-1$ to level j. The derivation now rests upon monitoring the error $(Z_k^n)^{-1}(I_k - D_k^n)\rho_k^n$ occurring in (5.22) through monitoring all errors $(I_j - D_j^n)\lambda_j^n, j \leqq k$, occurring in (6.3). The idea is to select the matrices D_j^n such that the error functions $(I_j - D_j^n)\lambda_j^n$ become sufficiently small. This makes sense because if C_3 and C_4 are stability constants such that

$$\|(Z_j^n)^{-1}\| \leqq C_3, \qquad \left\|\prod_{i=k-1}^{j+1} X_i^n\right\| \leqq C_4, \tag{6.5}$$

then

$$\|(Z_k^n)^{-1}(I_k - D_k^n)\rho_k^n\| \leqq C_3(1+\|P_{k-1k}\|(k-2)C_3C_4) \max_{2\leqq j\leqq k} \|(I_j - D_j^n)\lambda_j^n\|. \tag{6.6}$$

Hence, if for $k=2,\dots,l$, the matrices D_k^n are selected such that

$$C_3(1+\|P_{k-1k}\|(k-2)C_3C_4) \max_{2\leqq j\leqq k} \|(I_j - D_j^n)\lambda_j^n\| \leqq c\|(Z_k^n)^{-1}\tau D_k^n\alpha_k^n\|, \tag{6.7}$$

then the refinement condition (5.22) is satisfied.

In general, the stability constants C_3 and C_4 are unknown. However, if the dissipativity assumption (A5) is satisfied, then the constant $C_3 \leqq 1$. Furthermore, if we use linear interpolation, then (4.16) applies and C_4 also can be set equal to one, so that (6.7) simplifies to

$$\max_{2\leqq j\leqq k} \|(I_j - D_j^n)\lambda_j^n\| \leqq \frac{c}{k-1} \|(Z_k^n)^{-1}\tau D_k^n\alpha_k^n\|, \qquad k=2,\dots,l. \tag{6.8}$$

If assumption (A5) does not hold or if higher-order interpolation is used, then C_3 and C_4 may be larger than one, but not by a considerable amount. C_3 shall generally be of moderate size in view of the excellent stability behaviour of implicit Euler. Our practical experience with fourth-order Lagrangian interpolation is that higher-order interpolation is unlikely to yield instability problems, thus indicating that $\|X_k^n\|$, and hence C_4, are also of moderate size. That is why we proceed with (6.8) and also use it in situations where (A5) may be violated and/or higher-order interpolation is used.

In application, it suffices to impose (5.22) for $k=l$ only, so that (6.8) can be replaced by

$$\max_{2\leqq k\leqq l} \|(I_k - D_k^n)\lambda_k^n\| \leqq \frac{c}{l-1} \|(Z_l^n)^{-1}\tau D_l^n\alpha_l^n\|. \tag{6.9}$$

In order to satisfy this condition, estimates of λ_k^n have to be computed. Therefore, to create an extra safety margin, we replace (6.9) by the slightly more conservative condition

$$\|(I_k - D_k^n)\zeta_k^n\| \leqq \frac{c}{l-1} \|(Z_l^n)^{-1}\tau D_l^n\alpha_l^n\|, \qquad k=2,\dots,l, \tag{6.10}$$

where, componentwise, ζ_k^n is defined as

$$(\zeta_k^n)_i = |(\gamma_k^n)_i| + |(P_{k-1k}(Z_{k-1}^n)^{-1}\tau D_{k-1}^n\alpha_{k-1}^n)_i|. \tag{6.11}$$

Condition (6.10) will determine the integration domain of ω_k. Let Ω_k^n be this integration domain and recall that when a node belongs to Ω_k^n, the corresponding diagonal entry of D_k^n is equal to one and zero otherwise. Suppose that the maximal level number l and $c(l-1)^{-1}\|(Z_l^n)^{-1}\tau D_l^n\alpha_l^n\|$ are known and that a solution at Ω_{k-1}^n, $k\leqq l$, has just been computed. Prior to the integration step on level k, our task is then to determine Ω_k^n. That is, we must define D_k^n such that (6.10) is satisfied and in such a way that the area of Ω_k^n is as small as possible. The actual selection of Ω_k^n is carried out by a flagging procedure that scans level-k grid points. A point is flagged if, using appropriate estimates,

$$(\zeta_k^n)_i > \frac{c}{l-1} \|(Z_l^n)^{-1}\tau D_l^n\alpha_l^n\|. \tag{6.12}$$

Hence, for such a point the corresponding diagonal entry $(D_k^n)_{ii}=1$, and for nonflagged points we define $(D_k^n)_{ii}=0$. Thus the refinement condition (6.10) is satisfied.

In conclusion, the solution at a node of grid ω_k is interpolated only if a corresponding component of ζ_k^n is smaller than the maximum of the spatial discretization error at the finest grid multiplied with $\tau c(l-1)^{-1}$. Otherwise, integration is carried out at this node. No doubt this imposes a severe restriction on the size of the interpolation errors. On the other hand, this restriction is natural because when going to a higher level within the current time step, we never return to a grid point where the solution has been interpolated, which means that the interpolation error will be carried along to the next time step. The fact that we do not return is a direct consequence of the nesting property of the integration domains, which we will discuss next.

6.2. Restricted interpolation and the nesting property. We now introduce the nesting property of the integration domains. Recall that this property, being hidden in the definition of the matrices D_k^n, has played no role in the foregoing analysis. We stipulate that in application the nesting is enforced by the flagging procedure; in other words, this procedure scans only level-k points lying within the previous integration domain Ω_{k-1}^n. A direct consequence is that, unlike (3.5), the interpolation is carried out only for level-k points within Ω_{k-1}^n. Here we will justify the deviation due to this restricted interpolation. We will argue that the restricted interpolation is in fact allowed by the inequality (6.10), where interpolation over the whole of ω_k is still assumed.

Consider the error $(Z_{k-1}^n)^{-1}\tau D_{k-1}^n\alpha_{k-1}^n$, contained in ζ_k^n. This spatial error is defined at level $k-1$ and, by definition of D_{k-1}^n, has zero components outside Ω_{k-1}^n. Hence, all its prolongated components are taken into account in the flagging procedure for determining Ω_k^n. For the interpolation error γ_k^n, which lives on the whole of ω_k (grid expansion), the situation is different. However, restricted interpolation is allowed if for all level-k points outside Ω_{k-1}^n, the interpolation error satisfies

$$|(\gamma_k^n)_i| \leq \frac{c}{l-1}\,\|(Z_l^n)^{-1}\tau D_l^n \alpha_l^n\|, \tag{6.13}$$

because then points outside Ω_{k-1}^n will not be flagged if the interpolation step (3.5) would be carried out on the whole of ω_k. In other words, if (6.13) holds outside Ω_{k-1}^n, then the integration domains found with the restricted interpolation over Ω_{k-1}^n are equal to the domains found if the interpolation would be carried out on the whole of ω_k, which is in accordance with the method description (3.3a, b).

The following argument shows that inequality (6.13) is very plausible with the restricted interpolation procedure. First we recall that Ω_1^n coincides with the entire physical domain. Hence for $k=2$ there is no restricted interpolation, so that for all level-2 points outside Ω_2^n, inequality (6.13) is trivially satisfied. Next consider the case $k=3$. Now the interpolation is restricted to level-3 points within Ω_2^n. Since for all level-2 points outside Ω_2^n inequality (6.13) is satisfied, we are justified in supposing that this is also true for all level-3 points outside Ω_2^n, in view of the consistency of the interpolation (level-3 interpolation errors are smaller than level-2 errors). Further, by construction of Ω_3^n, (6.10) is satisfied for all level-3 points within Ω_2^n and outside Ω_3^n, and so is (6.16). In conclusion, we may suppose that (6.13) is satisfied for all level-3 points outside Ω_3^n when using the restricted interpolation for $k=3$. For $k=4$ and so on this argument can be repeated.

6.3. Implementation aspects. On top of the flagging procedure implementing (6.12), a safety measure has been built. Any node for which (6.12) is true is flagged together with its eight neighbours. Next, to create an extra buffer, all sides of cells with at least one flagged corner node are bisected. This means that a buffer zone of two mesh widths is used around any intolerable node. Near boundaries, physical and internal ones, the buffering differs slightly. Although in theory this buffering could be omitted, in practice it is wise to create a buffer zone around intolerable nodes because the estimation of higher spatial derivatives contained in α_k^n and γ_k^n is prone to inaccuracies. After the flagging procedure, a cluster algorithm groups all flagged nodes together to form the newly defined integration domain.

The parameter c in (6.12) must be specified. In view of result (5.25), c should be taken small so that the spatial accuracy obtained is indeed nearly equal to the spatial accuracy obtained without adaptation. In fact, the smaller c is, the more points will be flagged and hence the safer the local refinement will be ($c=0$ implies global refinement). On the other hand, when c is too large, it can occur that space errors are large and refinement is necessary but no nodes are flagged because (6.12) is satisfied at every node. Hence, c is available as a tuning parameter. In the experiments in § 7 we have simply put $c=1$.

Estimates of spatial interpolation and discretization errors are required. For $2 \leqq k \leqq l$ we must estimate the interpolation error γ_k^n and the prolongated spatial discretization error $P_{k-1k}(Z_{k-1}^n)^{-1}D_{k-1}^n\alpha_{k-1}^n$. Further, an estimate of the spatial discretization error $(Z_l^n)^{-1}D_l^n\alpha_l^n$ committed at the final lth level must be available at all lower levels. Because we use local uniform grids, the estimation of these errors can be realized cheaply and easily. Consider the error α_k^n (cf. (5.2)) and let p be the order of consistency (in this paper $p=2$). The estimation we apply is based on the use of a second spatial discretization operator $\tilde{F}$ of a higher-order $\tilde{p}$. After some elementary calculations we obtain the approximation

$$\alpha_k^n \approx \tilde{F}_k(t_n, u_k(t_n)) - F_k(t_n, u_k(t_n)) \tag{6.14}$$

as an asymptotically correct estimator for α_k^n. The benefit of using uniform grids now lies in the fact that $\tilde{F}$ is easily constructed. At internal nodes our $\tilde{F}$ provides fourth-order accuracy (standard symmetrical differences), while at nodes adjacent to physical or internal boundaries third-order accuracy is realized (standard one-sided differences). The benefit of using uniform grids is also reflected in the estimation for the error γ_k^n (cf. (5.4)). So far we have implemented Lagrangian interpolation of second (linear) and fourth order. For the second-order interpolation we need to estimate spatial derivatives u_{xx}, etc., while in the fourth-order case spatial derivatives like u_{xxxx} appear. For both cases the estimation is straightforward.

We emphasise that, in spite of its simplicity, linear interpolation may become disadvantageous due to the low order of accuracy. Inspection of the various terms in (6.12) suggests a comparison between the following order relations:

$$\tau(P_{k-1k}(Z_{k-1}^n)^{-1}D_{k-1}^n\alpha_{k-1}^n)_i = \mathcal{O}(\tau h_{k-1}^2), \tag{6.15a}$$

$$\|\tau(Z_l^n)^{-1}D_l^n\alpha_l^n\| = \mathcal{O}(\tau h_l^2), \tag{6.15b}$$

$$(\gamma_k^n)_i = \mathcal{O}(h_k^2), \quad \text{second-order Lagrangian (linear)}, \tag{6.15c}$$

$$(\gamma_k^n)_i = \mathcal{O}(h_k^4), \quad \text{fourth-order Lagrangian}. \tag{6.15d}$$

In the discretization terms the step size τ is contained. Consequently, it is the interpolation error that may govern the refinement if τ is very small, and particularly so when the interpolation is linear. The comparison is clearly in favour of the fourth-order interpolation.

To estimate the right-hand side term $\|(Z_l^n)^{-1}D_l^n\alpha_l^n\|$ of (6.12) for $2 \leqq k \leqq l-1$, we exploit the asymptotics. Since the mesh width of level k is half that of level $k+1$, we thus invoke

$$\|(Z_l^n)^{-1}D_l^n\alpha_l^n\| \approx 2^{-p(l-k)}\|(Z_k^n)^{-1}D_k^n\alpha_k^n\|, \qquad l \geqq k+1, \tag{6.16}$$

for $k = 1, 2, \ldots$. In theory it suffices to do this only for $k = 1$, but since for larger values of k this estimation will become more and more accurate, it is done for every k.

Finally, we will make a few remarks about the approximations (6.14) and (6.16). Our method, like every other adaptive grid method, is designed to solve PDEs with steep solutions. Yet (6.14) and (6.16) underlie asymptotics, which means that they are only accurate if the solution is sufficiently smooth on the grid in use. This constitutes a problem for LUGR methods, because these methods estimate errors on coarse grids. Nevertheless, if in practice the estimated error is not that accurate, it might still give a good indication of where the spatial error is large and where it is not and, specifically, the estimated error might still be in the same order of magnitude as the exact error, in which case the implemented refinement strategy based upon (6.14) will still work. In our experience so far, this is indeed the case. We believe this is due to the fact that estimation (6.16) is carried out for finer and finer subgrids with an increasing accuracy which partly remedies the problem. However, if solutions become very steep, it might be necessary to improve the implementation of the refinement condition (5.22).

7. Numerical example. This section is devoted to an illustration of the foregoing error analysis. Our goal here is to numerically illustrate that by imposing the refinement condition, the usual order behaviour is recovered. At the same time, the spatial accuracy obtained is comparable to the spatial accuracy on the finest grid if this grid would be used without adaptation.

7.1. The issue of implicitness. We use the implicit Euler method for time integration. In connection with implicitness, two points are worth mentioning. The first is that at any time step refinement takes place at different levels, resulting in a different Jacobian per level whose order usually varies. This impedes the profitable use of old Jacobians (like in sophisticated stiff ODE solvers), unless it is decided not to adapt grids at every time step, but instead per prescribed number of steps. We consider this as part of an overall strategy that can easily be placed on top of the existing one. We adapt grids at every time step since our main aim with the experiments is to illustrate the convergence analysis together with the refinement strategy. However, when dealing with real applications, it is most likely to be more advantageous to omit adaptation at every time step, just for efficiency reasons. The second point is that the Jacobians do not possess a regular band structure, since the integration domains Ω_k^n normally have an irregular shape. Unlike the first, this point is intrinsic to the local refinement method. In the experiments reported here, the Harwell sparse matrix solver MA28 has been used. This solver is well suited to coping with the structure we meet, but is rather time consuming for the present application. It is likely that standard iterative methods can be applied at lower costs.

7.2. The example problem. The problem is hypothetical and due to [1]. The equation is the linear parabolic equation

$$u_t = u_{xx} + u_{yy} + f(x, y, t), \quad 0 < x, y < 1, \quad t > 0, \tag{7.1}$$

and the initial function, the Dirichlet boundary conditions, and the source f are selected so that the exact solution is

$$u(x, y, t) = \exp[-80((x - r(t))^2 + (y - s(t))^2)], \tag{7.2}$$

where $r(t)=\frac{1}{4}[2+\sin(\pi t)]$ and $s(t)=\frac{1}{4}[2+\cos(\pi t)]$. This solution is a cone that is initially centered at $(\frac{1}{2},\frac{3}{4})$ and that symmetrically rotates around $(\frac{1}{2},\frac{1}{2})$ in a clockwise direction with a constant speed. We have used this problem to subdue our refinement method to a convergence test. Observe that the semidiscrete version of this problem satisfies the dissipativity assumption (A5).

7.3. Convergence experiments. We have carried out two identical convergence experiments. In the first, linear interpolation was used and in the second, fourth-order Lagrangian was used. In both the solution is computed four times over the interval $0\leqq t\leqq 2$, using a uniform 10×10 base grid and a constant time step size τ. In the first computation $l=1$, in the second $l=2$, and so on. Since per computation the smallest mesh width is halved, τ is simultaneously decreased by 2^2 in view of the first order of implicit Euler. Hence, in line with our analysis, per computation the maximal global error should also decrease by 2^2.

Table 7.1 shows the maxima of global errors restricted to the finest integration domain in use. This table also contains the maxima of the errors for the corresponding grid used without adaptation. The table clearly reveals the expected order behaviour. The errors of the $l=4$ runs are about a factor four smaller than the corresponding errors of the $l=3$ runs. Note that there is hardly a difference between the corresponding errors, showing that, as anticipated by our strategy, the choice of interpolant has no notable influence on the error. We emphasise that, in spite of the relatively large values for τ, the spatial error dominates the global errors shown in this table. For example, using $\tau=0.125$ instead of $\tau=0.5$ in the $l=2$ run, the same global errors are found (they deviate in the third or fourth decimal digit). In other words, conclusions on the spatial error behaviour induced by the local refinement algorithm can be drawn from this table. These results convincingly show, for the current example problem, that the use of the refinement condition ensures that the spatial accuracy obtained is very much comparable to the spatial accuracy on the finest grid if this grid is used without any adaptation. Finally we note that the choice $c=1$ apparently has no influence on the error. We owe this to the fact that the refinement condition has been derived from error bounds and is thus conservative.

The use of the two different interpolants is expressed in the slightly different integration domains shown in Figs. 7.1 and 7.2. As expected, at the higher levels linear

TABLE 7.1
Maxima of global errors restricted to the finest domain. Comparison with errors on a standard uniform grid.

				t			
τ	No. of levels	Interpolation	Single grid	0.50	1.00	1.50	2.00
2.00000	1		10×10				0.16447
0.50000	2	linear		0.03876	0.03890	0.03891	0.03891
		fourth order		0.03929	0.03945	0.03946	0.03946
			20×20	0.03865	0.03881	0.03882	0.03882
0.12500	3	linear		0.01369	0.01369	0.01369	0.01369
		fourth order		0.01376	0.01376	0.01376	0.01376
			40×40	0.01389	0.01389	0.01389	0.01389
0.03125	4	linear		0.00340	0.00340	0.00340	0.00340
		fourth order		0.00359	0.00359	0.00359	0.00359
			80×80	0.00347	0.00347	0.00347	0.00347

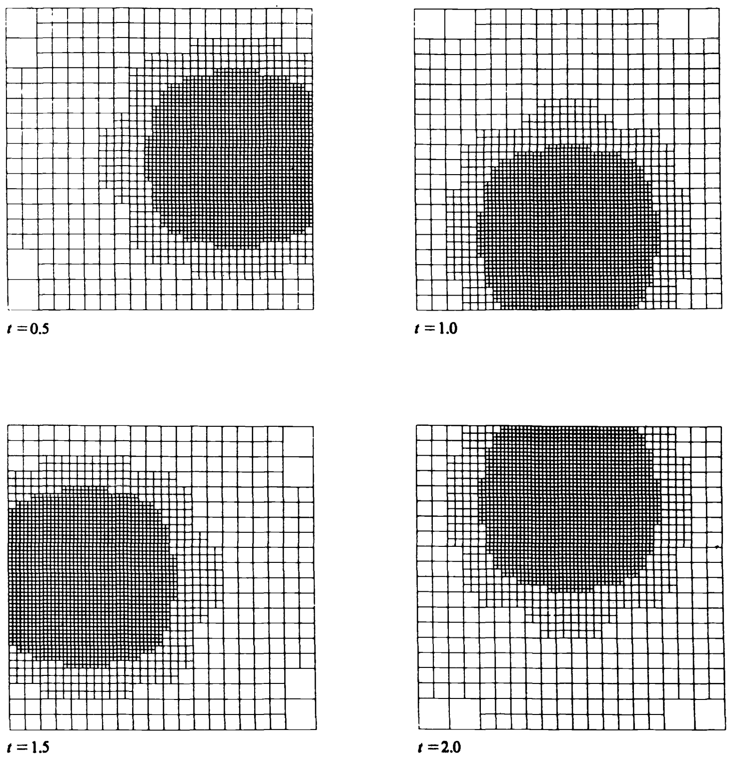


FIG. 7.1. *Linear interpolation. Integration domains for the $l = 4$ run at four different times. The size of the integration domains decreases only slowly with the number of levels. This is due to the fact that the cone is not very steep. At the end time,* $t = 2.0$, *the number of nodes amounts to* 121, 425, 813, *and* 1917, *respectively.*

interpolation gives rise to somewhat larger domains, showing that linear interpolation is more expensive. As a rule, fourth-order interpolation is to be preferred, as it leads to smaller domains. Note that for both interpolants the moving domains accurately reflect the symmetric rotation of the cone, which once again nicely illustrates the reliability of the implemented refinement condition with the various estimators.

8. Final remarks and future plans. In our future research we plan to pay more attention to time-stepping efficiency. Using the refinement strategy of this paper as a starting point, we plan to examine the application of methods possessing a higher order in time. Natural candidates belong to the class of Runge–Kutta methods. It should be stressed, though, that fully implicit methods can only be of serious advantage if the numerical algebra issue can be satisfactorily solved. In this connection splitting methods of the ADI and LOD type (see [9]) may therefore provide an attractive alternative to fully implicit ones, although they are usually less accurate in time. Another point of serious practical concern is to apply methods not only using an a priori chosen number of levels, but to also have the possibility to vary the number of levels. This might be useful for the computation of solutions that, for example, steepen in time, like the combustion problem in [13]. For such problems, the application of a variable number of levels should be combined with the use of variable temporal step sizes. Preferably, the complete adaptation should then be monitored by estimators of temporal and spatial errors in such a way that there is a balance between the two which aims at minimizing the waste of computing time.

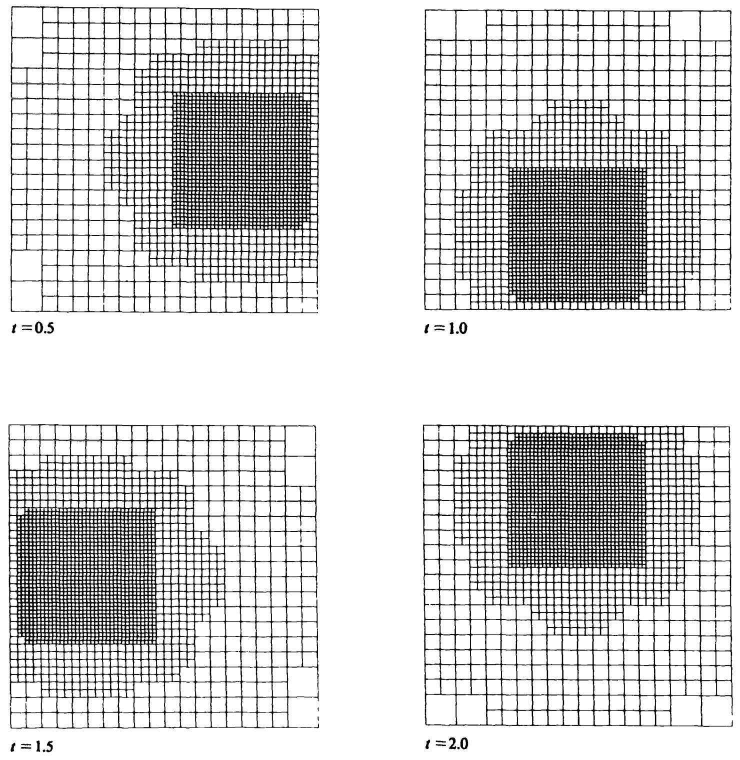


FIG. 7.2. *Fourth-order interpolation. Integration domains for the* $l = 4$ *run at four different times. At the end time,* $t = 2.0$, *the number of nodes amounts to* 121, 425, 813, *and* 1361, *respectively.*

Acknowledgment. We would like to thank Willem Hundsdorfer for his careful reading of the manuscript.

REFERENCES

[1] S. ADJERID AND J. E. FLAHERTY, *A local refinement finite element method for two-dimensional parabolic systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 792-811.

[2] D. C. ARNEY AND J. E. FLAHERTY, *An adaptive local mesh refinement method for time-dependent partial differential equations*, Appl. Numer. Math., 5 (1989), pp. 257-274.

[3] M. J. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484-512.

[4] K. DEKKER AND J. G. VERWER, *Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations*, North-Holland, Amsterdam, New York, Oxford, 1984.

[5] R. E. EWING, *Adaptive grid refinement for transient flow problems*, 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.

[6] W. D. GROPP, *A test of moving mesh refinement for 2D-scalar hyperbolic problems*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 191-197.

[7] ———, *Local uniform mesh refinement with moving grids*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 292-304.

[8] ———, *Local uniform mesh refinement on vector and parallel processors*, in Large Scale Scientific Computing, P. Deuflhard and B. Engquist, eds., Birkhäuser Series in Progress in Scientific Computing, Birkhaüser-Verlag, Basel, 1987, pp. 349-367.

[9] W. H. HUNDSDORFER AND J. G. VERWER, *Stability and convergence of the Peaceman-Rachford ADI method*, Math. Comp., 53 (1989), pp. 81-101.

[10] S. McCormick and J. W. Thomas, *The fast adaptive composite grid* (FAC) *method for elliptic equations*, Math. Comp., 46 (1986), pp. 439–456.

[11] J. E. Flaherty, P. K. Moore, and C. Ozturan, *Adaptive overlapping grid methods for parabolic systems*, 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.

[12] 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.

[13] R. A. Trompert and J. G. Verwer, *A static-regridding method for two dimensional parabolic partial differential equations*, Appl. Numer. Math., 8 (1991), pp. 65–90.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 279-294, March 1993

VECTOR-SUPERCOMPUTER EXPERIMENTS WITH THE PRIMAL AFFINE LINEAR PROGRAMMING SCALING ALGORITHM*

K. O. KORTANEK†

Abstract. The vector-supercomputer CRAY series has provided significant speed and significant digits accuracy for solving difficult, large-scale, and ill-conditioned linear programming problems with the linear programming (LP) primal scaling algorithm of Dikin [*Soviet Math. Dokl.*, 8 (1967), pp. 674-675]. Ranging from fully dense Chebyshev approximation-type matrices to highly sparse, relatively small, band-widths matrices generated from continuum mechanics problems, numerical results are presented that indicate good stability and objective function value accuracy. In a significant number of these experiments, a substantial speed-up in CPU time for obtaining good objective function accuracy has been obtained over a very stable implementation of the simplex method, termed LINOP. The companion QR-based scaling implementation, SHP, was applied to these dense problems, where high accuracy levels are required.

A conjugate gradient-based implementation, termed HYBY, was applied to a discretized plane strain plasticity problem for ill-conditioned problems having up to 7700 equations in as many as 9000 variables. The FORTRAN code includes an effective stability enhancement over the original affine scaling algorithm. The sparse LP matrix generator for these problems is available from the author (kort@icaen.uiowa.edu) or from Professor E. Christiansen in Odense, Denmark (edc@imada.ou.dk).

Key words. linear programming, affine scaling algorithm, QR decomposition, conjugate gradient method, semi-infinite programming problems

AMS(MOS) subject classifications. 65D15, 65F10, 65K05, 73E20, 90C05

1. Introduction. LP approximations to semi-infinite programs. Generally speaking, difficult to solve linear programs arise when linear semi-infinite programs are approximated using finite discretizations. Regardless of the kind of discretization used, one usually obtains a finite linear inequality subsystem of the original infinite one, which is potentially ill conditioned.

Building on the work of Gustafson [18] and Gustafson and Kortanek [19], an LP approximation was taken as Phase I in the semi-infinite programming codes of Fahlander [12]. In her codes, Phase I provides a starting solution for the necessary optimality conditions appearing in the form of a nonlinear system of equations. In yet another, more recent, direction a discretization method of Hettich [21] relies exclusively on discretizations, but in a new way that permits a substantial reduction in the number of inequalities in the finite linear subsystem.

Until now these linear programs have generally been solved by simplex method implementations which strive to achieve stability and high accuracy. For example, a rather recent stable simplex method code has been developed based on Georg and Hettich [15], and it has been tested on some difficult two-sided approximation problems and on some vibrating membrane problems.

Recently, another computational approach for linear programming has arisen, based on the interior point LP methods of Dikin [11] and Karmarkar [27]. In this

* Received by the editors August 5, 1987; accepted for publication (in revised form) March 21, 1992. This research was supported in part by National Science Foundation grant ECS-85-00940 and Office of Naval Research grant N00014-85-K-0198 at Carnegie-Mellon University and National Science Foundation grant ECS-86-96087 at the University of Iowa. Portions of this paper were presented at the First ICIAM Meeting held in Paris, June 29-July 3, 1987, and at a Mathematics Colloquium at the University of Trier, July 9, 1987. This paper is based on a working paper, 87-2 from the College of Business Administration, University of Iowa, Iowa City, which was distributed in May, 1987.

† Department of Management Science, University of Iowa, College of Business Administration, Iowa City, Iowa 52242-0001.

paper the Dikin algorithm, termed the LP affine scaling algorithm, is tested on some discretized semi-infinite programming problems. Two classes of problems are studied that for different reasons can be considered large scale.

The first class consists of problems that owe their size to either single-precision or double-precision, fully dense, constraint matrices. These problems are principally two-sided approximation problems (also termed Chebyshev or L_∞-norm approximation problems). Within this class we investigate the following problems: (a) Chebyshev approximation of two variable transcendental functions by polynomials in two variables (see Hettich [21] and Hettich and Zencke [26]); (b) complex variable Chebyshev approximation arising in antenna beam-forming problems (see Streit and Nuttall [36] and Streit [34], [35]); and (c) Chebyshev approximation problems that arise in computing eigenvalues of a vibrating membrane (see Hettich [22] and Hettich, Haaren, Ries, and Still [24]).

The second class of problems stems from computing limit collapse loads in a plastic plane strain problem, as studied in engineering continuum mechanics, and the reader is referred to Christiansen and Kortanek [9], [10] for model details and numerical results. Basically, a linear program is obtained from two different kinds of discretizations: (i) possibly different finite element approximations for the strain and stress forces of the problem, and (ii) an outer linear approximation of the unbounded convex yield set of the stresses.

All computational experiments reported here were performed on CRAY supercomputers: first on a CRAY 1 at AT&T's Murray Hill, NJ, facility, in the spring of 1986, and then on a CRAY X-MP/48 at the University of Illinois's Urbana-Champaign Supercomputer Center during 1987 and 1988.

These results are given in §§ 4 and 5. But first, in the next section, a proof of the affine scaling algorithm's asymptotic linear convergence rate is given under the rather mild assumptions of Kortanek and Shi [28]. This section also contains Hettich's suggestion for a stability improvement, which has been found to be essential for the plasticity application.

Section 3 describes the characteristics of the FORTRAN codes used in the numerical experiments.

2. The primal LP affine scaling algorithm and a convergence analysis. Consider a standard form of an LP problem:

(I) Let c be an n-vector, b an m-vector, and A a full-rank $m \times n$ matrix, none of whose columns is zero.

$$\text{Find } v_I = \min\{c^T x \mid x \in R^n,\ Ax = b,\ x \geqq 0\}. \tag{1}$$

It is natural to assume that the matrix A has the opposite sign property because it may be readily imposed without loss of generality using classical LP regularization procedures. The opposite sign property is equivalent to the constraint set of (1) being spanned by its extreme points (a characterization that also holds in infinite dimensions).

2.1. The primal affine scaling algorithm (Dikin [11]).

Step 0. Let $x^{(0)}$ be an (I)-feasible point, satisfying $x_i^{(0)} > 0$, for $i = 1, \ldots, n$. Let α be a scalar $0 < \alpha < 1$ and set the iteration count k to 0.

Step 1. Let $\bar{c}_k = D_k c$, $\bar{A}_k = AD_k$, where $D_k = \operatorname{diag}(x^{(k)})$. Set $y^{(k)} = (\bar{A}_k \bar{A}_k^T)^{-1} \bar{A}_k \bar{c}_k$ and $c_P^{(k)} = \bar{c}_k - \bar{A}_k^T y^{(k)}$.

Step 2. Test: if $c_P^{(k)} \neq 0$, then continue to Step 3, else stop with optimal $x^{(k)}$ for (I) and $y^{(k)}$ optimal for dual LP of (I).

Step 3. Let $(1/\gamma_k) = \max_i c_{Pi}^{(k)}$. Set $\hat{c}_P^{(k)} = \gamma_k c_P^{(k)}$.

Step 4. Set $x^{(k+1)} = x^{(k)} - \alpha D_k \hat{c}_P^{(k)}$, $k := k+1$, and return to Step 1.

In Step 1, $c_P^{(k)}$ is the orthogonal projection of $\bar{c}_k$ onto the null space of $\bar{A}_k$, where $y^{(k)}$ is also the unique solution of the least squares minimization (where only the two-norm is used in this paper)

$$\min_{y \in R^m} \|\bar{c}_k - \bar{A}_k^T y\|^2. \tag{2}$$

For an explanation and geometric motivation of this algorithm, the reader may consult Strang [33, pp. 683-686], where the algorithm is referred to as the "rescaling algorithm." In addition, one may consult Bazaraa, Jarvis, and Sherali [3, pp. 412-418] for a discussion of the "affine scaling variant of Karmarkar's algorithm."

Various elementary properties of the algorithm have appeared in an expanding literature. For definiteness, we state the following lemma from Kortanek and Shi [28].

LEMMA 1. *Assume that the algorithm does not terminate. Then*

(a) $\gamma_k \|c_P^{(k)}\| \geqq 1$ *for each* k;

(b) $\bar{c}_k^T c_P^{(k)} = \|c_P^{(k)}\|^2$ *for each* k;

(c) $\lim_k \gamma_k \|c_P^{(k)}\|^2 = 0$; *and*

(d) $\lim_k c_P^{(k)} = 0$.

Sufficient conditions for convergence usually require some form of primal or dual nondegeneracy assumption. Among the weakest of such conditions are those independently derived in Sherali, Sharpness, and Kim [32, p. 407] and Kortanek and Shi [28, Thm. 2, p. 52]. Theorem 1 in the latter reference can be used to establish an asymptotic linear convergence rate of the algorithm under a weakened nondegeneracy assumption.

THEOREM 1 (see [28]). *Define a set of coordinate positions by* $S_0 = \{i \mid \underline{\lim}_k x_i^{(k)} > 0\}$. *Assume that the column subset* $\{A^j \mid j \in S_0\}$ *has rank* m. *Then any limit point of* $\{x^{(k)}\}_k$ *solves* (I), *and any* $\{y^{(k)}\}_k$ *convergences to an optimal solution of the dual program of* (I).

The related convergence rate result is now the following.

THEOREM 2. *Assume that not every feasible point of* (I) *is optimal. Then under the assumptions of Theorem* 1 *and a nonterminating sequence* $\{x^{(k)}\}_{k=1}^{\infty}$, *the following inequality holds for sufficiently large* k:

$$\frac{c^T x^{(k+1)} - c^T \bar{x}}{c^T x^{(k)} - c^T \bar{x}} \leqq 1 - \frac{\alpha \gamma_k \|c_P^{(k)}\|}{2\sqrt{n}}.$$

Proof. Let $\bar{x}$ be a limit point of $\{x^{(k)}\}$. By Theorem 1, $\bar{x}$ is a solution to (I), $\lim_k y^{(k)}$ exists (say $\bar{y}$), and $\bar{y}$ solves the dual problem, i.e., $\bar{y}^T A^j \geqq c_j, j = 1, \ldots, n$, and $c^T \bar{x} = \bar{y}^T b$. From the definition of the projection $c_P^{(k)}$, we have

$$c_j - y^{(k)T} A^j = c_{P_j}^{(k)} / x_j^{(k)} \quad \text{for } j = 1, 2, \ldots, n.$$

Hence, by Lemma 1(d) and the definition of S_0 in Theorem 1, it follows that

$$\bar{y}^T A^i = c_i \quad \text{for each } i \in S_0.$$

Now S_0 is contained in the set $T_0 := \{j \mid c_j - \bar{y}^T A^j = 0\}$, which cannot equal $\{1, 2, \ldots, n\}$ because otherwise every (I)-feasible point is optimal. Hence we can rewrite $Ax^{(k)} = b$ as $\sum_{j \in T_0} A^j x_j^{(k)} + \sum_{j \notin T_0} A^j x_j^{(k)} = b$, yielding

$$\sum_{j \in T_0} \bar{y}^T A^j x_j^{(k)} + \sum_{j \notin T_0} \bar{y}^T A^j x_j^{(k)} = \bar{y}^T b.$$

With appropriate substitutions this becomes

$$\sum_{j \in T_0} c_j x_j^{(k)} + \sum_{j \notin T_0} \bar{y}^T A^j x_j^{(k)} = c^T \bar{x}. \tag{3}$$

It also follows that for sufficiently large k,

$$(4)\qquad c_j - y^{(k)T}A^j > \tfrac{1}{2}(c_j - \bar{y}^T A^j) \quad \text{for all } j \notin T_0.$$

This motivates adding $\sum_{j\notin T_0} c_j x_j^{(k)}$ to both sides of (3) and rearranging in order to obtain

$$(5)\qquad \sum_{j=1}^{n} c_j x_j^{(k)} - c^T\bar{x} = \sum_{j\notin T_0} (c_j - \bar{y}^T A^j) x_j^{(k)}.$$

Combining (4) with (3) gives

$$(6)\qquad c^T x^{(k)} - c^T\bar{x} < 2 \sum_{j\notin T_0} (c_j - y^{(k)T}A^j) x_j^{(k)}.$$

Applying the classical inequality

$$\sum_{i=1}^{n} a_i b_i \leqq \sqrt{n}\left(\sum_{i=1}^{n} a_i^2 b_i^2\right)^{1/2}, \qquad a_i b_i > 0,$$

yields

$$\sum_{j\notin T_0} (c_j - y^{(k)T}A^j) x_j^{(k)} \leqq \sqrt{n}\, \|c_P^{(k)}\|,$$

which combines with (5) to yield

$$c^T x^{(k)} - c^T\bar{x} < (2\sqrt{n})\|c_P^{(k)}\|.$$

Step 4 of the algorithm and the fact that $\bar{c}_k^T c_P^{(k)} = \|c_P^{(k)}\|^2$ (Lemma 1(b)) yields

$$c^T x^{(k+1)} = c^T x^{(k)} - \alpha\gamma_k \|c_P^{(k)}\|^2.$$

Because of nontermination, $c^T x^{(k)} - c^T\bar{x} > 0$ for each k, and hence we conclude that for sufficiently large k,

$$(7)\qquad \frac{c^T x^{(k+1)} - c^T\bar{x}}{c^T x^{(k)} - c^T\bar{x}} \leqq 1 - \frac{\alpha\gamma_k \|c_P^{(k)}\|}{2\sqrt{n}}. \qquad \square$$

Remark 1. The proof has similarities and has benefitted from the proof in Barnes [1]. A related convergence result was obtained by Ferris and Philpott [13, p. 275] in a more general context.

Remark 2. Since $\gamma_k\|c_P^{(k)}\| \geqq 1$ for each k, (7) establishes that the asymptotic convergence rate is linear. Empirically easy to observe, one might obtain insight on the speed of convergence by observing $\gamma_k\|c_P^{(k)}\|$, but we have not done this in a systematic way.

2.2. Obtaining an initial point (Phase I calibration procedure). Let μ be a positive number, and let M be the $n \times n$ diagonal matrix with μ along the diagonal. Let $\mathbf{e}$ denote the n-vector of all ones and set $\mathbf{s}^0$, $s_{n+1}^0 = (\mathbf{e}, 1)$. For $k = 0, 1, 2, \ldots$, apply just one iteration of the primal affine scaling algorithm to

$$\min_{\mathbf{s}, s_{n+1}} \quad s_{n+1}$$

$$\text{subject to } AM\mathbf{s} + \frac{(b - AM\mathbf{s}^k)}{s_{n+1}^k} s_{n+1} = b$$

$$\text{and} \quad \mathbf{s}, s_{n+1} \geqq 0,$$

beginning with the exactly feasible initial solution $\mathbf{s}^k$, s_{n+1}^k, and delivering the next iterate $\mathbf{s}^{k+1}$, s_{n+1}^{k+1}. Terminate when s_{n+1}^k is below a specified positive tolerance. Many people have had extensive computational experience with this procedure, beginning with Cavalier and Soyster [5].

In our experiments, we did no additional stability checks on the accuracy of the projection in Step 1 of the algorithm for Phase I.

Typically, setting $\mathbf{x}^{(0)}=\mathbf{s}^{(k)}$ for relatively small numbers of iterations k gave a sufficiently accurate interior point starting solution for application of the algorithm to the original LP problem. Typically, once a starting solution $\mathbf{x}^{(0)}$ is available, application of the algorithm is then termed "Phase II." In our experience, Phase II definitely requires checking whether the projection $c_P^{(k)}$ lies in the null space of $\bar{A}_k$ for each k, as described in § 3.

2.3. A stability enhancement. Although the projection $c_P^{(k)}$ of Step 1 converges to zero, each vector, $\bar{c}_k$ and $\bar{A}_k^T y^{(k)}$, employed in the difference expression of Step 1 may be relatively large.

During his visit to Carnegie-Mellon University in spring 1986, Hettich suggested another least squares problem to replace (2) in an attempt to "shorten" each of the vectors used in the difference expression, namely:

$$(8)\qquad \min_{y_\delta\in R^m} \|\bar{c}_k^{(\delta)}-\bar{A}_k^T y_\delta\|^2,$$

where $\bar{c}_k^{(\delta)}=\bar{c}_k-\bar{A}_k^T y^{(k-1)}$. Thus, at the current iteration k, the vector $\bar{c}_k$ of Step 1 is replaced by $\bar{c}_k^{(\delta)}$, which depends on the previous solution $y^{(k-1)}$. The solution to (8), denoted $y_\delta^{(k)}$, becomes

$$(9)\qquad y_\delta^{(k)}=(\bar{A}_k\bar{A}_k^T)^{-1}\bar{A}_k(\bar{c}_k-\bar{A}_k^T y^{(k-1)})=y^{(k)}-y^{(k-1)}.$$

Under the conditions of Theorem 1, it follows that $\bar{c}_k^{(\delta)}$ and $y_\delta^{(k)}$ both converge to 0 as k increases. Finally, observe that the projection in (8) is $c_P^{(k)}$ itself, i.e.,

$$\begin{aligned}\bar{c}_k^{(\delta)}-\bar{A}_k^T y_\delta^{(k)}&=\bar{c}_k-\bar{A}_k^T y^{(k-1)}-\bar{A}_k^T y^{(k)}+\bar{A}_k^T y^{(k-1)}\\&=\bar{c}_k-\bar{A}_k^T y^{(k)}=c_P^{(k)}.\end{aligned}$$

While we cannot generally say that the original vectors $\bar{c}_k$ and $\bar{A}_k^T y^{(k)}$ are large enough to cause stability problems in computing $c_P^{(k)}$, (8) permits the expression of $c_P^{(k)}$ as a difference of two vectors, each converging to zero.

For implementation purposes, the following modification of (8) is employed:

$$(10)\qquad \min_{y_\beta\in R^m} \|\bar{c}_k^{(\beta)}-\bar{A}_k^T y_\beta\|^2,$$

where $y_\beta^{(k)}$ denotes the optimal solution at stage k, beginning with $y_\beta^{(0)}=y^{(0)}=(\bar{A}_0\bar{A}_0^T)^{-1}\bar{A}_0\bar{c}_0$ as the initialization, and where $\bar{c}_k^{(\beta)}=\bar{c}_k-\bar{A}_k^T y_\beta^{(k-1)}$.

The solution $y_\beta^{(k)}$ to (10) is then

$$y_\beta^{(k)}=(\bar{A}_k\bar{A}_k^T)^{-1}\bar{A}_k(\bar{c}_k-\bar{A}_k^T y_\beta^{(k-1)})=y^{(k)}-y_\beta^{(k-1)},$$

and thus, unlike the solution to (8), $y_\beta^{(k)}$ here becomes a telescoping series. Again, (10) yields a projection onto the null space of $\bar{A}_k$, and analogous to (8), it is also $c_P^{(k)}$:

$$\begin{aligned}\bar{c}_k^{(\beta)}-\bar{A}_k^T y_\beta^{(k)}&=\bar{c}_k-\bar{A}_k^T y_\beta^{(k-1)}-\bar{A}_k^T(y^{(k)}-y_\beta^{(k-1)})\\&=\bar{c}_k-\bar{A}_k^T y^{(k)}=c_P^{(k)}.\end{aligned}$$

The enhancement was employed only in the sparse matrix implementation of the primal affine algorithm, which is described in the next section.

3. Features of the FORTRAN codes: SHP, LINOP, HYBY. LINOP is a simplex method implementation based on Georg and Hettich [15], while SHP and HYBY are implementations of the primal affine scaling algorithm of § 2.1. All codes are written in FORTRAN 77.

3.1. SHP. SHP was written by Miao Gen Shi of Tsinghua University for highly dense problems requiring a very high level of accuracy. The solution to the least squares problem is accomplished by a Householder transformation-based QR decomposition of the matrix $\bar{A}_k^T$ implemented with LINPACK subroutines

$$\bar{A}_k^T = Q^{(k)}R^{(k)}, \quad \text{where } Q^{(k)} \text{ is an } n \times m \text{ matrix, } Q^{(k)T}Q^{(k)} = I,$$

and

$$R^{(k)} \text{ is right upper triangular } m \times m.$$

As is well known, one can dispense with the $R^{(k)}$ factor in computing the projection, i.e.,

$$c_P^{(k)} = (I - Q^{(k)}Q^{(k)T})\bar{c}_k.$$

In SHP the relative accuracy is denoted by EP. From Lemma 1(b) we know ideally that $\bar{c}^T c_P = \|c_P\|^2$ (ignoring k sub- and superscripts). Because of round-off error, the relative error RS is not zero, where

$$\text{RS} = |\bar{c}^T c_P - \|c_P\|^2| / \|c_P\|^2.$$

RS provides a measure of stability loss and is used as one of the termination criteria. Iterations continue as long as RS is below a stability parameter choice, usually 1.00 or 0.10, i.e., the position of $c_P^{(k)}$ is acceptable, unless the norm of the projection is sufficiently small, in which case the algorithm is also terminated.

As an additional termination check on the accuracy of the projection, we typically provide $\|Ax^{(k)} - b\|$ in the ∞- or two-norm.

We shall use the common FORTRAN edit descriptors e or d to record various constants in single or double precision. Thus, for example, $\text{e}-6 = 1.\text{e}-6 = 1.\text{d}-6 = 10^{-6}$.

For CRAY double-precision runs the choice of EP ranges from d − 17 to d − 21 (machine accuracy d − 28). For single-precision runs EP ranges from e − 12 to e − 13 (machine accuracy e − 14). The uniform choice of α in Step 0 of the algorithm is 0.999 for both phases, which could be termed an attempt at taking "long steps" in the descent directions. Smaller α leads to more iterations, with about the same accuracy.

3.2. LINOP. Full details of this FORTRAN code appear in "the package LINOP" by Hettich and collaborators. Georg and Hettich [15] have shown that the most stable simplex method implementation occurs when the orthogonal Q-matrix is retained and itself updated through successive iterations. See also Hettich and Zencke [25].

In LINOP the relative accuracy is denoted by EPS, where, for example, EPS = 1.e − 14 means that single-precision numbers are accurate up to 14 decimal places. EPS determines when a number should be set to 0, or when a QR factorization is inaccurate. EPS is also used to detect a nearly singular basis matrix. The code outputs error indications when any of these states are encountered.

Both LINOP and SHP were compiled on CRAY compilers available at Murray Hill, NJ, and at Urbana-Champaign, IL, which provide for a certain amount of automatic vectorization. But no other specific vectorization coding techniques were used in these basically serially written FORTRAN codes. Important possibilities for speed-up obtained by removing inhibitors to vectorization are reported in Zenios and Mulvey [42].

Because both of these codes retain the orthogonal factors in the matrix decompositions, comparisons between their performances should show them to be equal. While LINOP has a self-correcting Q-updating procedure and SHP does not, one might still expect similar performance in CPU time because of the many more simplex iterations needed, in contrast to the greatly reduced number of iterations of the scaling algorithm.

This has been confirmed in numerical experiments for classes of Chebyshev approximations problems.

3.3. HYBY. In early 1986, Kortanek wrote a FORTRAN code for a sparse implementation of the algorithm using the classical conjugate gradient algorithm for solving (10); see [20, Algorithm 4]. Two clear advantages of this approach are: (1) $\bar{A}_k$ can be stored in its sparse form, never having to compute $\bar{A}_k\bar{A}_k^T$, and hence saving on storage; and (2) one can use the current $y_\beta^{(k)}$, the solution to (10), for an initial, advanced start in the next iteration's least squares problem (10).

Analogous to various sparse matrix packages (for example, Paige and Saunders [30]), we use three arrays for the storage of $\bar{A}_k^T$. Array JMA lists the nonzero elements of $\bar{A}_k^T$, row by row. Array IM indicates the column number of each entry in JMA. Finally, array MM gives the number of nonzero entries in each row of $\bar{A}_k^T$.

The matrix $\bar{A}_k$ is preconditioned using a diagonal matrix with entries formed from the two-norm of the rows of $\bar{A}_k$. Other more effective preconditioners related to a Cholesky factorization of $\bar{A}_k\bar{A}_k^T$ are discussed, for example, in Georg and Liu [16].

HYBY employs the same stability check and stopping rules that SHP does. Typically, for single-precision CRAY runs, a cut-off for the norm of the residuals for the inner loop conjugate gradient iterations is taken to be about 1.e − 13. Relaxing this criterion gives projections that do not lie sufficiently within the null space of $\bar{A}_k$. HYBY is clearly less accurate than SHP, and so we always include a post-optimality check of the minimum of the "reduced costs," i.e.,

$$\min\{c_j - A_j^T y^{(k)} \mid j = 1, \dots, n\}, \tag{11}$$

ideally nonnegative, where A_j is the jth column of A. At this point we simply omit post-duality checking in SHP because of its rather high accuracy.

Our experience on CRAY machines is that running in double precision can result in increases in CPU time by a factor of at least 10. Therefore, it is essential to run HYBY in single precision when applied to large-scale LP problems.

We describe next the results of some numerical experiments.

4. Numerical experiments on vector-supercomputers.

4.1. Two-sided approximation by polynomials in two dimensions: SHP and LINOP with double precision. Historically, the task of solving finite LP approximations to infinite problems set in a continuum led to the development of LP codes, such as LINOP [15], having a high degree of accuracy when double precision is used.

A typical two-sided approximation problem can be viewed as a semi-infinite program and further discretized.

(A) Given (continuous) functions $f(x), u_1(x), \dots, u_n(x)$ defined on a set S in R^2 compute

$$v_P = \inf_{y_1,\dots,y_n,y_{n+1}} y_{n+1}$$

$$\text{s.t.} \quad \left|\sum_{i=1}^n y_i u_i(x) - f(x)\right| - y_{n+1} \leqq 0 \quad \text{for all } x \in S.$$

Removing the absolute value signs leads to the equivalent problem

$$v_P = \inf y_{n+1}$$

$$\text{s.t.} \quad \sum_{i=1}^n y_i u_i(x) + y_{n+1} \geqq f(x),$$

$$-\sum_{i=1}^n y_i u_i(x) + y_{n+1} \geqq -f(x) \quad \text{for all } x \in S.$$

The approximation problem to be solved will be obtained by discretizing the set S, i.e., selecting a finite subset T of S, and then constructing the dual to the above minimization program:

$$\text{(B)} \qquad \sup_{\lambda^+(x),\lambda^-(x)} \sum_x f(x)\lambda^+(x) - \sum_x f(x)\lambda^-(x)$$

$$\text{s.t.} \quad \sum_x \begin{pmatrix} u_1(x) \\ \vdots \\ u_n(x) \\ 1 \end{pmatrix} \lambda^+(x) - \sum_x \begin{pmatrix} u_1(x) \\ \vdots \\ u_n(x) \\ 1 \end{pmatrix} \lambda^-(x) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix},$$

and all $\lambda^+(x)$, $\lambda^-(x) \geqq 0$, and where "$\sum_x$" denotes "$\sum_{x\in T}$."

Typically one can use a linear independence argument, which depends on properties of the u_i-functions and the mesh of the grid S to show that the constraint matrix in (B) has full row rank. It clearly has the opposite sign property because of the last equation. Observe that the constraint matrix is $(n+1)$ by $2 * (\# T)$, and that an interior point feasible solution is immediate; namely, assign $1/(2 * (\# T))$ to each variable in (B) where "#" means "the number of members of." Thus, applying SHP to (B) does not require a Phase I procedure. In the following problems, the full rank assumption holds for sufficiently fine uniform grids. For the first two experiments, note that if the polynomial has degree d, then the number of equations in (B) is $0.5(\mathrm{d}+2)(\mathrm{d}+1)+1$, as Tables 1 and 3 illustrate. In each case, the two-dimensional grids are simply uniform ones based on Cartesian products of respective one-dimensional grids.

TABLE 1
CRAY X-MP/48 *double-precision runs on* LP(B) *for approximating the function* x^y: *machine precision* 1.d−28.

Problem size			LINOP: Simplex algorithm			Scaling algorithm	
d	m	n	Phase I	Phase II	CPU sec.	Phase II	CPU sec.
5	22	288	136	41	7.5	11	9.8
6	29	288	249	63	17.4	11	15.7
7	37	288	451	82	38.9	12	26.0
7	37	338	488	102	47.6	11	28.0
7	37	392	457	98	49.4	12	35.1
7	37	450	516	130	62.5	14	45.8

TABLE 2
Comparisons of objective function values for the runs of Table 1.

Problem size (m, n)	LINOP: Simplex algorithm	Scaling algorithm
22,288	.1498601e−3	.1498411e−3
29,288	.1137433e−4	.1136960e−4
37,288	.9638871e−6	.9638175e−6
37,338	.9598364e−6	.9594715e−6
37,392	.9570501e−6	.9567180e−6
37,450	.9688830e−6	.9688692e−6

TABLE 3
CRAY X-MP/48 *double precision runs on* LP(B) *for approximating the function* $\log(x+y)\sin y$: *machine precision* 1.d−28.

Problem size			LINOP: Simplex algorithm			Scaling algorithm	
d	m	n	Phase I	Phase II	CPU sec.	Phase II	CPU sec.
5	22	288	107	57	7.0	15	12.6
6	29	288	237	124	19.9	15	20.5
7	37	288	422	163	43.1	13	27.9
7	37	338	444	196	51.8	14	34.4
7	37	392	435	216	58.0	15	42.7
7	37	450	508	207	69.6	16	51.6

In these experiments we test the codes SHP and LINOP on the dual problem (B) and provide results on the number of iterations, CPU time, and objective function values. In general, no further comparisons were made regarding primal or dual optimal solutions to (B) delivered by each code, but objective function value comparisons are given in Tables 2 and 4.

4.1.1. Experiment 1. Approximate x^y by

$$\sum_{i=0}^{d}\sum_{j=0}^{i} a_{ij}x^jy^{i-j}$$

on the square $[1,2]\times[1,2]$.

For the six problems counted in the order of increasing matrix size, the scaling precisions EP paired with the norms $\|Ax^{(k)}-b\|_\infty$ are, respectively, (d−19, 0.44e−15), (d−19, 0.44e−15), (d−19, 0.77e−15), (d−21, 0.41e−15), (d−21, 0.11e−14), (d−23, 0.23e−13). For LINOP the relative accuracy was chosen between d−19 and d−21.

For the 37×288 problem the two-norm of the difference of the dual vectors delivered by LINOP and SHP was about 0.90d−5. Dual vector components ranged from −0.96e−6 to 1.40 with most less than 1 in absolute value.

4.1.2. Experiment 2. Approximate $\log(x+y)\sin x$ by

$$\sum_{i=0}^{d}\sum_{j=0}^{i} a_{ij}x^jy^{i-j}$$

on the rectangle $[\varepsilon,1]\times[1,2.5]$, where $\varepsilon=1.\text{d}-5$. One could have also chosen $\varepsilon=0$ if desired.

TABLE 4
Comparisons of objective function values for the runs on Table 3.

Problem size (m,n)	LINOP: Simplex algorithm	Scaling algorithm
22,288	.15657066e−3	.1565698e−3
29,288	.3833631e−4	.3833208e−4
37,288	.9714904e−5	.9711184e−5
37,338	.9892332e−5	.9888688e−5
37,392	.9852445e−5	.9850761e−5
37,450	.9780506e−5	.9780050e−5

The experience on accuracy was similar to Experiment 1, except slightly more stable. For the scaling runs, EP was d − 19 for all but the largest run, where it was d − 21. The maximum norm $\|Ax^{(k)} - b\|_\infty$ ranged from .11e − 14 to .44e − 15.

4.2. Two-sided approximations for computing membrane eigenvalues: SHP and LINOP. We present next some numerical experiments on some two-sided approximation problems which arise in Hettich's parametric programming approach for computing eigenvalues of the problem $\Delta u = \lambda u$ in a region R with $u = 0$ on its boundary (see Hettich [22], Hettich, Haaren, Ries, and Still [24], and Hettich and Zencke [26]). In these experiments, for instance, R is an ellipse, and the LP problems are fully dense and rather ill conditioned. The LP preprocessors were written by Hettich and his colleagues. The computer runs were made in the spring of 1986 on AT&T's CRAY 1 at the Murray Hill, NJ, facility. Solution times were not recorded, merely iteration counts, and we tested SHP's Phase I procedure; see Table 5.

4.2.1. Experiment 3. In each problem the objective function values (OFVs) agreed up to seven figures. For the first two problems, the correct optimal bases were indicated by the scaling method with solution-component accuracy of three figures. For the third problem, a support set of 36 variables was indicated that had 32 variables in common with the 35-point support set returned by the simplex method. There is therefore strong reason to believe that the scaling method has indicated a nonextreme point optimal solution, having seven-digit OFV accuracy.

4.3. Complex function approximation for antenna array processing. Recent work in complex function approximation and its applications have appeared in Glashoff and Roleff [17]; Streit and Nuttall [36], [37]; Streit [34], [35]; and Watson [41] (see also Barrodale and Phillips [2]). We now present a prototype problem.

TABLE 5
CRAY 1 *double-precision solutions to selected membrane eigenvalue problems* (1986). *Number of iterations and some precision statistics.*

Problem size		LINOP:			Scaling algorithm			
Number of equations	Number of variables	Phase I	Phase II	EPS	Phase I	Phase II	$\|Ax - b\|_\infty$	EP
20	100	43	24	d − 19	2	17	0.35e − 19	d − 21
30	400	93	313	d − 19	2	20	0.39e − 15	d − 21
35	500	147	758	d − 19	2	37	0.22e − 14	d − 22

Let A_j denote the jth column of an $n \times m$ complex matrix A and let f be a complex m-vector, having components f_j. The unconstrained L_∞ complex approximation problem is then

$$(12)\quad \text{(CI)} \qquad \min_{z \in C^n} \max_{1 \leqq j \leqq m} |z^T A_j - f_j|,$$

where the vector of unknown z is taken to be a row vector, and "$|\cdot|$" is the complex absolute value.

A linearization of the nonlinear problem (CI) is based on the fact that

$$(13) \qquad |x + iy| = \max_{0 \leqq \theta < 2\pi} (x \cos \theta + y \sin \theta).$$

In order to obtain the tightest approximation based upon (13), a set of angles is selected according to

$$\theta_l = \pi(l-1)/p, \qquad l = 1, 2, \dots, 2p, \tag{14}$$

where $p \geqq 2$. We say that the *number of phase-shifted values* in an approximation based on (13) is $2p$. The integer p is thereby only *half* of them. Using this approximation problem, (CI) becomes

$$(\mathrm{CI}_p) \qquad \min_{z \in C^n} \max_{\substack{1 \leqq j \leqq m \\ 1 \leqq l \leqq 2p}} (z^T A_j - f_j)_{\theta_l},$$

where

$$\begin{aligned}(z^T A_j - f_j)_{\theta_l} &= (z^T A_j - f_j)^R \cos\theta_l + (z^T A_j - f_j)^I \sin\theta_l \\ &= z^R(A_j^R \cos\theta_l + A_j^I \sin\theta_l) + z^I(A_j^R \sin\theta_l - A_j^I \cos\theta_l) \\ &\quad - (f_j^R \sin\theta_l + f_j^I \sin\theta_l), \quad \text{a real quantity,}\end{aligned}$$

and where "R" and "I" denote real and imaginary parts of a complex quantity. Because of the choice of the θ_l's in (14), we see that

$$(z^T A_j - f_j)_{\theta_{p+l}} = (z^T A_j - f_j)_{(\pi + \theta_l)} = -(z^T A_j - f_j)_{\theta_l}.$$

Therefore, we may rewrite (CI_p) as an ordinary, two-sided, real approximation similar to the problems in § 4.1:

$$(\mathrm{CI}_p) = \min_{(z^R, z^I) \in R^{2n}} \max_{\substack{1 \leqq j \leqq m \\ 1 \leqq l \leqq p}} |(z^T A_j - f_j)_{\theta_l}|, \tag{15}$$

where the absolute value taken in (15) is just the real absolute value function.

As before, the primal LP problem associated with (15) will have $2n+1$ variables and $m(2p)$ linear inequalities. The dual LP problem, therefore, has $2n+1$ equations in $2pm$ variables. The current form of SHP will require taking (-1) times its objective function since SHP is a minimizer. Typically, for example, M could be 101 or 501. Typical numbers of full phase angle shifts yielding interesting problems could begin with 4, but recently Streit [34] has considered a class of problems with 1024 angle shifts.

A slight modification of the Streit and Nuttall [36] matrix generator, termed ZMAINS, generates the data for the following problem; see Streit [35].

In this application, a basis function is again of the form $\exp(ikx)$ over the interval $[u_0, 2-u_0]$, where k is a positive integer and where $u_0 = 0.0538117$. The function to be approximated is $f(x) = \exp(i49x)$. However, not all of the integers k from 1 to 48 are included in the list of basis functions. Some integer indices are to be omitted, specifically those in $\{7, 17, 21, 29\}$.

The experiments were conducted for approximating $f(x)$ by using various subsets of the full set of basis functions determined as follows.

Specify an integer N, $7 \leqq N \leqq 48$. Define $\{k_i\}_{i=1}^t$ to be $\{1, 2, \dots, N\} \backslash \{7, 17, 21, 29\}$, ordered according to $1 < k_1 < k_2 < \cdots < k_t$. Setting $\mathrm{NDEL} = \#\{1, 2, \dots, N\} \cap \{7, 17, 21, 29\}$ yields $t = N - \mathrm{NDEL}$. For example, if $N = 48$ and $M = 501$, then A is a complex matrix having 44 rows and 501 columns. If, in addition, there are $2p$ phase angle shifts, then the LP dual problem associated with (CI_p) of (15) has $2mp$ variables in $(2(N - \mathrm{NDEL}) + 1)$ equations.

The experiments reported below were done in single precision, which on the CRAY X-MP/48 gives approximately 14 digits. The decision on accuracy was based on the large, fully dense matrices generated for this problem. The numerical results are presented in Tables 6 and 7, where the initial start of § 4.1 was used for SHP.

It is our experience that EPS = 1.e − 12 is not sufficiently accurate for LINOP, and the small EPs of Table 6 are needed.

There is a loss of right-hand side accuracy with SHP compared to the approximation problems of §§ 4.1 and 4.2. $\|Ax^{(k)} - b\|_\infty$ ranges over (integer) * (10^{-6}) for the first five problems and (integer) * (10^{-5}) for the last three (and largest problems).

5. Computing the collapse state in material plastic limit analysis. During the last five years Christiansen and Kortanek have been applying sparse implementations of Dikin's [11] algorithm to a difficult class of LP problems arising in plastic collapse analysis. An earlier version of this manuscript contained preliminary results; see Beasley [4], which used the HYBY implementation of § 3. These results were improved using finer discretizations in Christiansen and Kortanek [9]. In 1989, they replaced the classical Hestenes and Stiefel conjugate gradient implementation with the highly accurate Paige and Saunders [30] least squares algorithm, and many variations of a two-dimensional plane strain problem were tested. In all of these experiments over all this time, Christiansen and Kortanek found that the stability enhancement of § 2 was essential.

TABLE 6
CRAY X-MP/48 *single precision complex approximation of a problem in Streit and Nuttall* [36].

Problem size		Simplex LINOP method iterations					Scaling algorithm		
N	M	$2P$	Phase I	Phase II	Time	EPS	Phase II	Time	EP
23	101	16	484	779	86.8	e − 14	18	17.4	e − 12
24	101	16	617	566	86.0	e − 14	17	18.1	e − 12
24	201	16	611	699	174.0	e − 14	19	25.0	e − 12
25	101	16	602	1186	132.9	e − 15	19	19.9	e − 12
25	201	16	612	1140	239.1	e − 15	22	27.7	e − 12
25	301	16	598	1350	384.0	e − 15	22	42.0	e − 12
26	301	16	687	713	293.8	e − 15	20	42.8	e − 12
27	301	16	804	839	354.2	e − 16	22	44.5	e − 13

TABLE 7
Comparisons of objective function values for the runs of Table 6.

Problem size (m, n)	Simplex algorithm	Scaling algorithm
41,1616	.9991863	.9991917
43,1616	.9991208	.9991729
43,3216	.9997221	.9997448
45,1616	.9991228	.9991854
45,3216	.9996893	.9997271
45,4816	.9998010	.9998146
47,4816	.9996448	.9997786
49,4816	.9994667	.9997870

The LP problems are constructed from a FORTRAN matrix generator that is easy to use. We give a description below so that others may also test their LP codes and compare their results with what we have obtained with the LP primal affine scaling algorithm.

The LP model has been put into the standard form (1) with the help of some slack variables based upon a discretization of the following plastic collapse problem. Consider a cube of a plastic material. In the y-x plane, make thin symmetric cuts $\frac{2}{3}$ of the distance to the center. Subject the cube to a uniform tensile force in the x-direction at the two surfaces given by $x=1$ and $x=-1$ (see Fig. 1) and increase this tensile force until the material collapses due to excessive interior stresses. The details are explained in Christiansen and Kortanek [9].

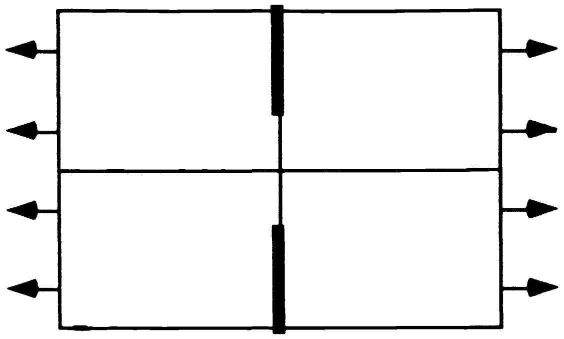


FIG. 1

The program takes as input the text file INDATA containing only one integer N, which must be a multiple of 3. $1/N$ is the element size in the finite element discretization. Varying N through values 3, 6, 9, ... will generate LP problems of increasing size. The outputs from the program are the data for LP (1) in sparse matrix form given in the three files discussed in § 3.3.

This test problem has a structure that is typical for problems in mechanics, but different from other LP test problems. The equality $\mathbf{Ax}=\mathbf{b}$ contains information on both the discrete equilibrium equation of the solid and on the inequalities (including slack variables) expressing the bounds on the stresses in the material.

It must be emphasized that not only the optimal value, but also primal and dual optimal vectors are important parts of the solution. There is evidence to the fact that at least the primal solution (representing the collapse stresses in the material) is not unique. In our experience Simplex codes cannot solve this problem as well as the LP primal affine scaling algorithm.

For large N the recent results obtained in Christiansen and Kortanek [10] were obtained using a modified version of the matrix generator. As it turned out, many of the linear constraints had positive slacks in the optimal solution obtained with the scaling algorithm. These constraints were eliminated and only checked a posteriori for the solution. This modification has not been possible when using Simplex codes. Insisting on extreme points apparently will introduce more active constraints "than necessary," when the optimal solution is nonunique. This modified set-up program is available upon request. Also available is a post-processor, which will extract fields for stresses and plastic flow from the primal and dual solutions and produce plots on a postscript printer.

6. Conclusions. All of the linear programs tested in this paper were constructed from discretizations of linear optimizations set in a continuum. Only the LP primal affine algorithm was tested on these problems during the five-year period beginning in 1985. An enormous amount of algorithmic development has occurred during this period, and a state-of-the-art survey on computer implementations was presented by Professor David Shanno at the SIAM National Meeting in Chicago in July, 1990.

In the numerical experiments reported in this paper, good convergence of objective function values was obtained within 20 Phase II iterations of the scaling algorithm. Whenever a Phase I routine was done, usually not more than three or four iterations were required. There were 30 computer runs reported in these experiments, on problems which can be considered large scale. Twenty-three of these were on fully dense Chebyshev approximation problems that are ill conditioned and require a large amount of storage. While the stable LP code LINOP provides the most accurate solutions, the scaling code SHP provides good approximations to the objective function values with significant speed-up in CPU time. For the membrane problem, there were additional checks on the accuracy of the optimal solutions themselves, and the results were favorable. In one membrane problem run, SHP indicated the presence of alternate optima.

No case was found where one code (LINOP or SHP) failed and the other did not. Typically, when a failure did occur, then ill-conditioning was experienced in SHP, while in LINOP an error message indicated that the QR factorization was inaccurate or that a nearly singular basis matrix had been detected. For the complex approximation problems, there was significant objective function value sensitivity to the choice of the relative precision parameters, EPS for LINOP and EP for SHP.

It is rather surprising that the large-scale results on these Chebyshev problems are "images" of earlier results obtained by solving much smaller Chebyshev approximation problems using PC equipment as well as an IBM 3083 mainframe at Carnegie-Mellon University. The number of Phase II iterations for these smaller problems were reported at the M.I.T. Mathematical Programming Symposium in August, 1985. They are quite consistent with the findings of the 23 experiments reported here.

Due in part to the immediate, interior point starting solution to the two-sided Chebyshev approximation problem, it can safely be asserted that the scaling algorithm returns a very good approximation to the objective function value at a significant speed-up over the simplex method.

Equally exciting are the results from applying HYBY to the highly sparse LP problem generated by the plane strain plastic limit analysis problem. The LP formulation is known to be highly degenerate with alternate optima. Over 15 years ago, Christiansen found that this was a difficult problem to solve because the extreme point solution delivered by a simplex method implementation depended upon the computer and simplex code used. It was therefore difficult to make inferences about the collapse fields of the true material whose extremal principal is formulated over a continuum.

Rather than presenting model details, we have referred the reader to Christiansen and Kortanek [9], [10] and made reference to the LP matrix generator, which the reader may obtain from either author. We have found the "stability enhancement" to be essential for these plasticity approximations, but perhaps readers and users may go directly to their own implementations of interior point methods to test their results against ours. We expect CPU speed-ups of at least a factor of 10 from using recently developed interior point methods. More essential for the understanding of material collapse, in our opinion, are the collapse fields themselves, which will require both primal and dual optimal linear programming solutions for particular discretizations.

Acknowledgments. This work greatly benefitted from discussions with R. Hettich during his sabbatical visit to Carnegie-Mellon University in the spring of 1986, and also from the earlier visits of Miao Gen Shi, E. Christiansen, and M. Vlach in 1985.

REFERENCES

[1] E. R. BARNES, *A variation on Karmarkar's algorithm for solving linear programming problems*, Math. Programming, 36 (1986), pp. 174–182.

[2] I. BARRODALE AND C. PHILLIPS, *Solution of an overdetermined system of linear equations in the Chebyshev norm algorithm* 495, ACM Trans. Math. Software, 1 (1975), pp. 264–270.

[3] M. S. BAZARAA, J. J. JARVIS, AND H. D. SHERALI, *Linear Programming and Network Flows*, 2nd ed., John Wiley, New York, 1990.

[4] J. E. BEASLEY, *Linear programming on CRAY supercomputers*, J. Oper. Res. Soc., 4 (1990), pp. 133–139.

[5] T. M. CAVALIER AND A. L. SOYSTER, *Some computational experience and a modification of the Karmarkar algorithm*, ISME Working Paper 85-105, The Pennsylvania State University, University Park, PA, 1985.

[6] E. CHRISTIANSEN, *Limit analysis for plastic plates*, SIAM J. Math. Anal., 11 (1980), pp. 514–522.

[7] ———, *Computation of limit loads*, Internat. J. Numer. Methods Engrg., 17 (1981), pp. 1547–1570.

[8] ———, *On the collapse solution in limit analysis*, Arch. Rational Mech. Anal., 91 (1986), pp. 119–135.

[9] E. CHRISTIANSEN AND K. O. KORTANEK, *Computing material collapse displacement fields on a CRAY X-MP/48 by the LP primal affine scaling algorithm*, Ann. Oper. Res., 22 (1990), pp. 355–376.

[10] ———, *Computation of the collapse state in limit analysis using the LP primal affine scaling algorithm*, J. Comput. Appl. Math., 34 (1991), pp. 47–63.

[11] I. I. DIKIN, *Iterative solution of problems of linear and quadratic programming*, Soviet Math. Dokl., 8 (1967), pp. 674–675.

[12] K. FAHLANDER, *Computer programs for semi-infinite optimization*, TRITA NA-7312, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, S-10044, Stockholm, Sweden, 1973.

[13] M. C. FERRIS AND A. B. PHILPOTT, *An interior point algorithm for semi-infinite programming*, Math. Programming, 43 (1989), pp. 257–276.

[14] K. GEORG, *A numerically stable update for simplicial algorithms*, in Numerical Solution of Nonlinear Equations, E. Allgower, K. Glashoff, and H.-O. Peitgen, eds., Springer Lecture Notes in Mathematics 878, Springer-Verlag, New York, Berlin, 1981, pp. 117–127.

[15] K. GEORG AND R. HETTICH, *On the numerical stability of the simplex algorithm*, Universität Trier, Trier, Germany, 1985.

[16] J. A. GEORG AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[17] K. GLASHOFF AND K. ROLEFF, *A new method for Chebyshev approximation of complex-valued functions*, Math. Comp., 36 (1981), pp. 233–239.

[18] S.-Å. GUSTAFSON, *On the computational solution of a class of generalized moment problems*, SIAM J. Numer. Anal., 7 (1970), pp. 343–357.

[19] S.-Å. GUSTAFSON AND K. O. KORTANEK, *Numerical treatment of a class of semi-infinite programming problems*, Naval Res. Logist. Quart., 20 (1973), pp. 477–504.

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

[21] R. HETTICH, *An implementation of a discretized method for semi-infinite programming*, Math. Programming, 34 (1986), pp. 354–361.

[22] ———, *On the computation of membrane-eigenvalues by semi-infinite programming methods*, in Infinite Programming, E. J. Anderson and A. B. Philpott, eds., Lecture Notes in Economics and Math. Systems 259, Springer-Verlag, Berlin, Heidelberg, New York, 1985, pp. 79–89.

[23] R. HETTICH AND G. GRAMLICH, *A note on an implementation of a method for quadratic semi-infinite programming*, Math. Programming, 46 (1990), pp. 249–254.

[24] R. HETTICH, E. HAAREN, M. RIES, AND G. STILL, *Accurate numerical approximations of eigenfrequencies and eigenfunctions of elliptic membranes*, Z. Angew. Math. Mech., 67 (1987), pp. 589–597.

[25] R. HETTICH AND P. ZENCKE, *Numerische methoden der approximation und semi-infiniten optimierung*, Teubner Mathematik, Stuttgart, 1982.

[26] ———, *Two case-studies in parametric semi-infinite programming*, in Systems and Optimization, A. Bagchi and H. Th. Jongen, eds., Springer Lecture Notes Contr. and Inf. Sciences 66, Springer-Verlag, New York, Berlin, 1985, pp. 132–155.

[27] N. K. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373-395.

[28] K. O. Kortanek and M. G. Shi, *Convergence results and numerical experiments on a linear programming hybrid algorithm*, European J. Oper. Res., 32 (1987), pp. 47-61.

[29] G. Opfer, *Solving complex approximation problems by semi-infinite optimization techniques: A study on convergence*, Numer. Math., 39 (1982), pp. 411-420.

[30] C. C. Paige and M. A. Saunders, *ALGORITHM* 583 LSQR: *Sparse linear equations and least squares problems*, ACM Trans. Math. Software, 8 (1982), pp. 195-209.

[31] H. D. Sherali, *Algorithmic insight and a convergence analysis for a Karmarkar type of algorithm for linear progamming problems*, Naval Res. Logist. Quart., 34 (1987), pp. 399-416.

[32] H. D. Sherali, B. O. Skarpness, and B. Kim, *An assumption-free convergence analysis for a perturbation of the scaling algorithm for linear programs, with application to the L_1 estimation problem*, Naval Res. Logist. Quart., 35 (1988), pp. 473-492.

[33] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, MA, 1986.

[34] R. L. Streit, *ALGORITHM* 635: *An algorithm for the solution of systems of complex linear equations in the l_∞ norm with constraints on the unknowns*, ACM Trans. Math. Software, 11 (1985), pp. 242-249.

[35] ———, *Solutions of systems of complex linear equations in the l_∞ norm with constraints on the unknowns*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 132-149.

[36] R. L. Streit and A. H. Nuttall, *A general Chebyshev complex function approximation procedure and an application to beam forming*, J. Acoust. Soc. Amer., 72 (1982), pp. 181-190.

[37] ———, *A note on the semi-infinite programming approach to complex approximation*, Math. Comp., 40 (1983), pp. 599-605.

[38] T. Tsuchiya, *Global convergence property of the affine scaling methods for primal degenerate linear programming problems*, Math. Oper. Res., 17 (1992), pp. 527-557.

[39] R. J. Vanderbei, M. S. Meketon, and B. A. Freedman, *A modification of Karmarkar's algorithm*, Algorithmica, 1 (1986), pp. 395-407.

[40] R. J. Vanderbei and J. C. Lagaris, *I. I. Dikin's convergence result for the affine-scaling algorithm*, in Mathematical Developments Arising from Linear Programming: Proceedings of a Joint Summer Research Conference, Bowdoin College, Brunswick, ME, June/July 1988, J. C. Lagarias and M. J. Todd, eds., Contemp. Math., 114, American Mathematical Society, Providence, RI, 1990, pp. 109-119.

[41] G. A. Watson, *Numerical methods for Chebyshev approximation of complex-valued functions*, in Algorithms for Approximation II, J. C. Mason and M. G. Cox, eds., Chapman and Hill, London, New York, 1990, pp. 246-264.

[42] S. A. Zenios and J. M. Mulvey, *Nonlinear network programming on vector supercomputers: A study on the CRAY XMP*, Oper. Res., 34 (1986), pp. 667-682.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 295-309, March 1993

NUMERICAL COMPUTATION OF THE SCATTERING FREQUENCIES FOR A CYLINDRICALLY SYMMETRIC POTENTIAL*

MUSHENG WEI† AND GEORGE MAJDA‡

Abstract. In this paper, a novel numerical algorithm is described for computing the scattering frequencies (also called resonances, resonance poles, or poles of the scattering matrix) of a bounded, cylindrically symmetric potential with compact support in $\mathbb{R}^3$. The potential can be time independent, or it can depend periodically on time. The algorithm is based on a characterization of the scattering frequencies as the exponents in the long-time asymptotic expansion of the solution of the initial value problem for the time-dependent scalar wave equation perturbed by the potential. It uses a discretization of the initial value problem for the time-dependent wave equation perturbed by the potential combined with a procedure for extracting the exponents from the computed solution. A number of numerical computations are also presented to document the performance and reliability of the algorithm.

Key words. scattering frequencies, perturbed wave equation, potential, spectral method, least squares

AMS(MOS) subject classifications. 65N35, 65L06

1. Introduction.

Scattering frequencies (SFs), also called resonances, resonance poles, or poles of the scattering matrix, form a discrete set of complex constants which occur in many scattering problems. Important examples include acoustic wave scattering by a bounded potential or in the exterior of a bounded obstacle, and electromagnetic wave scattering in the exterior of a dielectric or conducting body. In each of these problems, the SFs characterize the local rates of decay of a reflected wave, and thereby contain information about the scatterer (potential, obstacle, dielectric, conducting body). This fact is used by engineers and physicists to try to recover properties of a (usually unknown) scatterer from the SFs. See [1], [2], [5], [9], [13], and [14] for details.

Unfortunately, in each of the stated scattering problems, the precise relationship between a scatterer and its scattering frequencies is not completely understood. Furthermore, SFs can be explicitly determined in very few problems. In this paper, we develop a novel numerical algorithm for computing SFs. Our ultimate goal is to use the results of the calculations to help unravel the true relationship between a scatterer and its scattering frequencies (see [18] and [28]).

In this paper, we first consider scattering by a real-valued, compactly supported, time-independent potential $q(x)$ defined for $x \in \mathbb{R}^3$. We assume throughout this paper that $q(x) \in L^\infty(\mathbb{R}^3)$ and vanishes for $r = |x|_2 > R$ with $R < \infty$. ($L^\infty(\mathbb{R}^3)$ denotes the set of essentially bounded functions; $|\cdot|_2$ denotes the Euclidean norm.) Then we consider the case when a potential depends periodically on time.

The scattering frequencies of $q(x)$ can be characterized in several equivalent ways. See [28, § 4] for a short summary and [13] for a detailed explanation. We will state the two characterizations that are appropriate for our discussion.

* Received by the editors May 7, 1990; accepted for publication (in revised form) March 30, 1992. The numerical computations for this research were supported by the National Science Foundation Supercomputer Center, Pittsburgh, Pennsylvania, under grant PSCA353, by the Supercomputer Center at the University of Minnesota, Minneapolis, and by the Ohio State Supercomputer Center under grant PAS330.

† Department of Mathematics, East China Normal University, Shanghai 200062, People's Republic of China. This author's work was supported by the Education Committee of the People's Republic of China.

‡ Department of Mathematics, The Ohio State University, Columbus, Ohio 43210-1174. This author's work was supported by Office of Naval Research grants N00014-87-K-0677 and N00014-88-K-0351.

The usual definition of a scattering frequency is a complex number σ for which there is a function $p(x)$ (not identically zero), called a *scattering eigenfunction*, such that

$$-\Delta p(x)+q(x)p(x)=\sigma^2 p(x), \tag{1.1}$$

$$p(x)\to \exp(-i\sigma r)/r \quad \text{as } r=|x|_2\to\infty. \tag{1.2}$$

Condition (1.2) is the classical outgoing radiation condition. This definition characterizes σ as a kind of eigenvalue. However, σ is not a true eigenvalue because $p(x)$ is not square integrable when $\operatorname{Im}\{\sigma\}>0$.

To formulate the second characterization of the SFs, we consider the initial value problem

$$u_{tt}-\Delta u+q(x)u=0 \quad \text{for } x\in\mathbb{R}^3, \quad t>0, \tag{1.3}$$

$$u(x,0)=f(x), \qquad u_t(x,0)=g(x), \tag{1.4}$$

where $f(x)$ and $g(x)$ have compact support. It can be shown (see [13] and [15]) that if

$$E(u(x,0))=\int_{\mathbb{R}^3}\left((g(x))^2+\sum_{i=1}^{3}(f_{x_i})^2\right)dx<\infty,$$

then for each fixed $x\in\mathbb{R}^3$, the solution of (1.3), (1.4) has a near-field expansion given by

$$u(x,t)\sim\sum_{j=1}^{\infty}c_j p_j(x)\,e^{i\sigma_j t} \quad \text{as } t\to\infty, \tag{1.5}$$

which is valid in the local energy norm. The constants c_j depend on the initial data, the functions $p_j(x)$ depend only on the potential and spatial location x (except in the case of multiple σ_j when powers of t may also occur), and the complex constants σ_j, the scattering frequencies, depend on the potential, but they are independent of the initial data and spatial location x. In fact, the functions $p_j(x)$ that appear in (1.5) are the solutions of (1.1), (1.2) with $\sigma=\sigma_j$.

A number of mathematical results have been proved about the SFs of a bounded compactly supported potential $q(x)$. In this introduction we state only those results that are relevant to our discussion.

For general (real) potentials $q(x)$, it is known that there are, at most, a countable number of complex SFs σ_j that are symmetric with respect to the imaginary axis and satisfy $\operatorname{Im}\{\sigma_j\}\to+\infty$ as $j\to\infty$ (see [13], [16], and [20]). Consequently, we can order the SFs by the relation $\operatorname{Im}\{\sigma_j\}\leqq\operatorname{Im}\{\sigma_{j+1}\}$. (The SFs σ_j, $j=1,\dots,N$ are called the first N leading SFs.)

More precise information is available for radially symmetric potentials $q(r)$ with $r=|x|_2$. Let (r,θ,φ) denote the usual spherical coordinates of a point in $\mathbb{R}^3$. Look for solutions of (1.3) with the form

$$u(x,t)=u(r,\theta,\phi,t)=e^{i\sigma t}h(r)Y_e^m(\theta,\varphi),$$

where $Y_e^m(\theta,\varphi)$ is a spherical harmonic of order l, $l=0,1,2,\dots$, and $|m|\leqq l$. Then $h(r)$ satisfies

$$-\frac{d^2h(r)}{dr^2}-\frac{2}{r}\frac{dh(r)}{dr}+\left(q(r)+\frac{l(l+1)}{r^2}\right)h(r)=\sigma^2h(r), \tag{1.6}$$

$$h(r)\to\exp(-i\sigma r)/r \quad \text{as } r\to\infty.$$

Newton (see [19] or [20, Chap. 12]) shows that the resonances (SFs) of $q(r)$ are precisely the values $\sigma_{l,j}$, $l=0,1,2,\dots,j=1,2,\dots$, such that (1.6) has a nontrivial solution $h(r)=h_{l,j}(r)$ when $\sigma=\sigma_{l,j}$. The SFs $\sigma_{0,j}$, $j=1,2,\dots$ are called the *radial scattering frequencies* of $q(r)$, while the remaining SFs of $q(r)$ are called *nonradial scattering frequencies.*

Despite the extensive work by mathematicians and physicists on the SFs of a potential, the properties of SFs are not completely understood. Since scattering frequencies can be explicitly determined only for very special potentials, it is important to develop reliable numerical algorithms to compute them.

There are several ways to numerically compute the SFs of a given potential. One method consists of discretizing the elliptic equation (1.1) with an appropriate boundary condition at the edge of a truncated domain. This procedure is discussed and applied in [6], [7], [22], and [23]. These authors are primarily concerned with long-range potentials which depend on the relative distances between several interacting particles. The methods in these papers have not been applied to potentials that have a nonradial spatial dependence, and they cannot be used when the potential depends periodically on time. A second method consists of rewriting (1.1), (1.2) as an appropriate integral equation and approximating the integral equation. Unfortunately, like the first procedure, this method cannot be used for potentials which depend periodically on time. A third method is based on the characterization of the SFs as the exponents in the long-time asymptotic expansion (1.5) of the solution of the initial value problem for (1.3). In a previous paper with Strauss [28], we used this characterization of the SFs to compute a finite number of leading radial SFs of some time-independent and periodically oscillating radially symmetric potentials. In this paper, we show how to modify the numerical algorithm in [28] in order to compute some nonradial SFs of a radially symmetric potential, and some SFs of potentials with cylindrical symmetry. The modified algorithm can be used to compute SFs for time-independent potentials and potentials that depend periodically on time.

Our numerical algorithm for computing the SFs of a potential consists of several steps. First, we choose a spatial location x and estimate a *finite time* $t_\infty(x)$, such that the asymptotic expansion (1.5) approximates the solution of (1.3), (1.4) for $t \geqq t_\infty(x)$. A formula for $t_\infty(x)$ based on the range of influence of the initial data is discussed in § 2.

In the second step, we solve the initial value problem (1.3), (1.4) using any convenient (but nontrivial) initial data with compact support. The numerical procedure can be any appropriate method such as a finite difference, Galerkin, or pseudospectral method. The main constraint on the numerical method is that it must produce an extremely accurate solution so that the SFs can be computed from the numerical solution. In § 3 we present a pseudospectral method for solving problem (1.3), (1.4) when the potential is cylindrically symmetric. In § 4 we discuss appropriate choices for the initial data (1.4), and in § 5 we explain the need for an extremely accurate numerical solution of problem (1.3), (1.4).

In the third step of our algorithm, we calculate the exponents in (1.5) from the numerical solution of (1.3), (1.4) evaluated at location x and at equally spaced times. To explain this procedure, set $t=t_k=t_\infty(x)+k\Delta T$, $u_k=u(x,t_\infty(x)+k\Delta T)$ for $k=1,2,\dots$, and $z_j=\exp(i\sigma_j\Delta T)$, $j=1,2,\dots$. Then the asymptotic expansion (1.5) approximates the exact solution and takes the form

$$u_k \approx \sum_{j=1}^{\infty} a_j(z_j)^k \quad \text{for } k=0,1,2,\dots,$$

where the a_j's are constants defined by $a_j = c_j p_j(x) \exp(i\sigma_j t_\infty(x))$. Therefore, in step 3 we calculate the z_j's from the u_k's. Of course, only a finite number of z_j's can be calculated. We have already discussed this delicate problem in [17], [26], and [27]. Our method implements a modified and improved version of a classical method due to Prony [21], which reduces the problem of computing the z_j's from the u_k's to solving overdetermined systems of linear equations (by least squares) and then finding roots of polynomials. In § 5 we discuss our version of Prony's method and the need for a highly accurate numerical solution of (1.3), (1.4).

In the last step of our algorithm, we check the accuracy of the results by repeating steps 1-3 using different sampling locations x with the same initial data. Since the mathematical theory of SFs shows that they are independent of the sampling location, the computed SFs should repeat when steps 1-3 are applied at different sampling locations.

In § 6 we present some numerical results that demonstrate the accuracy and reliability of our algorithm for computing nonradial SFs of time-independent radial potentials and potentials with cylindrical symmetry. We also consider potentials that depend periodically on time.

In § 7 we summarize the conclusions of our work and list other important scattering problems where our algorithm can be used.

Notation. The cylindrical coordinates (ρ, θ, z) of a point $(x, y, z) \in \mathbb{R}^3$ are defined by

$$x = \rho \cos\theta, \quad y = \rho \sin\theta, \quad z = z.$$

The spherical coordinates (r, θ, φ) of a point $(x, y, z) \in \mathbb{R}^3$ are defined by

$$x = r \sin\varphi \cos\theta, \quad y = r \sin\varphi \sin\theta, \quad z = r \cos\varphi.$$

2. Choice of $t_\infty(x)$.

It might seem difficult (if not impossible) to use the characterization of the SFs as the exponents in (1.5) to compute them. This characterization requires a long-time computation with a potentially large accumulation of roundoff and truncation errors which would destroy the accuracy of the computations. In [28] we presented an estimate for $t_\infty(x)$, the approximate time when (1.5) is valid at location x, when the potential $q(x)$ in (1.3) is radially symmetric. The argument in [28], based on Huyghen's principle for the wave equation in three dimensions, also applies when the potential is not radially symmetric.

If $q(x) \equiv 0$ for all $x \in \mathbb{R}^3$, the solution of (1.3), (1.4) propagates along the characteristics of the wave equation emanating from the initial data. Furthermore, if the supports of $f(x)$ and $g(x)$ are contained in a ball centered at the origin with finite radius R, then Huyghen's principle implies that for any $x \in \mathbb{R}^3$, the solution of (1.3), (1.4) satisfies $u(x, t) \equiv 0$ for $t \geqq |x|_2 + R$. Therefore, location x feels all of the influence of the initial data in the finite time interval $[0, |x|_2 + R]$.

If $q(x) \not\equiv 0$, then the solution of (1.3), (1.4) still propagates along the characteristics of the wave equation emanating from the initial data. Therefore, location x feels the direct influence of the initial data in the finite time interval $0 \leqq t \leqq |x|_2 + R$, and the residual influence of the initial data when $t > |x|_2 + R$. Consequently, we choose $t_\infty(x) = |x|_2 + R$. This choice is reasonable from the point of view of theory, and actual calculations on problems with exact solutions support this choice.

3. Numerical approximations of the perturbed wave equation.

In this section we describe a pseudospectral method to approximate the initial value problem (1.3), (1.4) when the potential q and the initial data are cylindrically symmetric. Before discussing the numerical method we make some important remarks.

Remark 1. We are able to determine a finite time $t_\infty(x)$ where (1.5) is approximately valid, so we only need to solve the initial value problem (1.3), (1.4) for a finite time interval $[0, T]$ with $T < \infty$ to compute the SFs from the numerical solution.

Remark 2. Let $B(0, R) = \{x \in \mathbb{R}^3 \colon |x|_2 \leqq R\}$. If the potential $q(x)$ and the initial data $f(x)$ and $g(x)$ have their supports contained in $B(0, R)$ with $R < \infty$, then Huyghen's principle implies that the solution of (1.3), (1.4) satisfies $u(x, t) \equiv 0$ for $|x| \geqq R + t$. Therefore, if we set $D = \{(x, t)\colon |x| \leqq R + T \text{ and } 0 \leqq t \leqq T\}$, then the initial value problem (1.3), (1.4) can be equivalently rewritten as the initial boundary value problem

$$u_{tt} - \Delta u + q(x)u = 0, \quad \text{for } (x, t) \in D, \tag{3.1}$$

$$u(x, 0) = f(x), \qquad u_t(x, 0) = g(x), \tag{3.2}$$

$$u(x, t) \equiv 0 \quad \text{for } |x| = R + T, \quad 0 \leqq t \leqq T. \tag{3.3}$$

This reformulation is used to reduce the amount of storage required to implement the numerical methods, and to suggest appropriate basis functions for the pseudospectral method.

Remark 3. Our numerical algorithm can be used to compute SFs of potentials $q(x)$ with $x \in \mathbb{R}^3$. However, due to computer storage limitations, we consider only the case when the potential and the initial data have cylindrical symmetry, that is, $q = q(\rho, z), f = f(\rho, z)$, and $g = g(\rho, z)$. Under this assumption the solution u of (3.1)-(3.3) is independent of θ, i.e., $u = u(\rho, z, t)$, and satisfies

$$u_{tt} - \left(u_{\rho\rho} + \frac{1}{\rho} u_\rho + u_{zz}\right) + q(\rho, z)u = 0 \tag{3.4}$$

$$\text{for } 0 < t \leqq T, \quad 0 \leqq \rho \leqq R + T = a, \quad |z| \leqq a,$$

$$u(\rho, z, 0) = f(\rho, z), \qquad u_t(\rho, z, 0) = g(\rho, z), \tag{3.5}$$

$$u_\rho(0, z, t) = 0 \quad \text{(regularity condition)}, \tag{3.6}$$

$$u(\rho, z, t) \equiv 0 \quad \text{for } \rho = a, \quad z = \pm a, \quad \text{and} \quad 0 \leqq t \leqq T. \tag{3.7}$$

We first tried to discretize problem (3.4)-(3.7) by finite difference methods in three different ways. A straightforward approximation of (3.4) using central difference quotients does not work due to the coordinate singularity at $r = 0$. We tried to modify this scheme by using a special averaging formula at the origin in the manner described by Strikwerda [30, pp. 386-387]. This method also did not produce an accurate solution. Finally, we approximated problem (3.4)-(3.7) using an idea due to Lapidus [29]. The idea is to solve the three-dimensional wave equation in Cartesian coordinates (to avoid the $1/r$ singularity) on an appropriate subset of grid points using the standard central difference approximation, and then to use the imposed symmetry and an interpolation procedure to generate an approximation to $u(\rho, \theta, t)$ at the other necessary grid points. This method produced reasonable results (see [26]) but the results were not as accurate as the ones obtained using the pseudospectral collocation method, which we will describe. Furthermore, this finite-difference procedure required a significantly longer CPU time to solve (3.4)-(3.7) than the pseudospectral method.

To discretize problem (3.4)-(3.7) by the pseudospectral method [10] or [25], we approximate $u(\rho, z, t)$ by

$$u_{N,M}(\rho, z, t) = \sum_{i=1}^{N} \sum_{j=1}^{M} b_{ij}(t) J_0(\lambda_i \rho / a) \sin(\pi j (z + a)/2a), \tag{3.8}$$

where $J_0(x)$ is the Bessel function of order zero, $a = R + T$, and λ_i, $i = 1, \dots, N+1$ are the first $(N+1)$ zeros of $J_0(x)$, which are arranged in increasing order so that $\lambda_{i+1} > \lambda_i$, $i = 1, \dots, N$. We use the Bessel functions $J_0(\lambda_i \rho/a)$ in the ρ-direction in order to eliminate the coordinate singularity at $\rho = 0$ in (3.4), and to make the numerical approximation $u_{N,M}(\rho, z, t)$ vanish at the boundary $\rho = a$. We use the functions $\sin(\pi j(z+a)/2a)$ in the z-direction because the exact solution of (3.4) satisfies $u(\rho, \pm a, t) = 0$ for $t < T$. Furthermore,

$$\frac{d^p u(\rho, \pm a, t)}{\partial z^p} = 0 \quad \text{for } p = 0, 1, 2, \dots \quad \text{and} \quad \frac{d^p}{dx^p} \sin(x)\bigg|_{x = \pm \pi j} = 0 \quad \text{for } p \text{ even},$$

so the choice of basis functions preserve some additional properties of the exact solution.

We use the collocation points given by

$$\text{(3.9)} \quad \rho_k = a\lambda_k/\lambda_{N+1}, \quad k = 1, \dots, N \quad \text{and} \quad z_l = a(2l/(M+1) - 1), \quad l = 1, \dots, M.$$

The collocation points z_l, $l = 1, \dots, M$ are equally spaced in the z-direction and, since $\lambda_k \approx (k - \frac{1}{4})\pi$, $k = 1, \dots,$ the collocation points ρ_k, $k = 1, \dots, N$ are nearly equally spaced in the ρ-direction. These facts permit us to use an explicit scheme with a reasonable time step when we discretize (3.4) in time (see (3.20)).

Equations for the unknowns $b_{ij}(t)$ are determined by substituting (3.8) into (3.4), evaluating the resulting expression at each collocation point (ρ_k, z_l), and using the definition of $J_0(x)$. This process yields the system of ordinary differential equations

(3.10)

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \left(\frac{d^2}{dt^2} b_{ij}(t) + b_{ij}(t)(\lambda_i/a)^2 + b_{ij}(t)(j\pi/2a)^2 + b_{ij}(t) q(\rho_k, z_l) \right)$$
$$\times J_0(\lambda_i \lambda_k/\lambda_{N+1}) \sin(jl\pi/(M+1)) = 0 \quad \text{for } k = 1, \dots, N \quad \text{and} \quad l = 1, \dots, M.$$

System (3.10) can be rewritten in a concise way by introducing appropriate matrices. Set

$$U_{NM}(t) = (u_{NM}(\rho_k, z_l, t))_{N\times M},$$
$$J_N = (J_0(\lambda_i \lambda_k/\lambda_{N+1}))_{N\times N},$$
$$S_M = \sqrt{2/(M+1)}\,(\sin(jl\pi/(M+1)))_{M\times M},$$
$$B(t) = (b_{ij}(t))_{N\times M}, \qquad \Lambda_N = \operatorname{diag}((\lambda_1/a)^2, \dots, (\lambda_N/a)^2),$$
$$\Sigma_M = \operatorname{diag}((\pi/2a)^2, \dots, (M\pi/2a)^2),$$
$$V_{NM}(t) = (q(\rho_k, z_l) \cdot u_{NM}(\rho_k, z_l, t))_{N\times M},$$

then J_N is symmetric, S_M is orthogonal, i.e., $S_M^{-1} = S_M^T = S_M$, and $U_{NM}(t) = J_N B(t) S_M$. Furthermore, if J_N is invertible (see Remark 4), then

$$B(t) = J_N^{-1} U_{NM}(t) S_M^{-1} = J_N^{-1} U_{NM}(t) S_M,$$

and (3.14) can be rewritten as

$$\text{(3.11)} \quad \begin{aligned} \frac{d^2}{dt^2} U_{NM}(t) &= -(J_N \Lambda_N J_N^{-1} U_{NM}(t) + U_{NM}(t) S_M \Sigma_M S_M + V_{NM}(t)) \\ &= F(U_{NM}(t)). \end{aligned}$$

The initial conditions for (3.11), obtained from (3.5), are given by

$$\text{(3.12)} \qquad U_{NM}(0) = (f(\rho_k, z_l))_{N\times M} \quad \text{and} \quad \frac{dU_{NM}(0)}{dt} = (g(\rho_k, z_l))_{N\times M}.$$

Remark 4. We have been unable to prove that J_N is invertible for all values of N. However, we have numerically computed the eigenvalues and the condition numbers for J_N for $N \leqq 200$. In all cases, we have found that J_N is invertible, and that the condition number $K = \|J_N\|_2 \|J_N^{-1}\|_2 < AN^\alpha + B$, where $\alpha < 0.6$, A and $B < 10$, and $\|\cdot\|_2$ denotes the matrix norm subordinate to the usual Euclidean norm.

We approximate the second-order initial value problem (3.11), (3.12) by the stable variant of Stoermer's rule introduced by Henrici [12] combined with two steps of active Richardson extrapolation. An excellent discussion of this method is presented in [8]. Specifically, let $h > 0$ denote a time step, let $\tilde{v}^n$ denote the numerical approximation of $v_{NM}(nh)$, and set $\Delta_k = \tilde{v}^{k+1} - \tilde{v}^k$. The stable variant of Stoermer's rule is defined by

$$\Delta_0 = h\left(\dot{v}_{NM}(0) + \frac{h}{2} F(v_{NM}(0))\right), \qquad \tilde{v}^1 = v_{NM}(0) + \Delta_0, \tag{3.13}$$

$$\Delta_k = \Delta_{k-1} + h^2 F(\tilde{v}^k), \quad \tilde{v}^{k+1} = \tilde{v}^k + \Delta_k, \quad k = 1, \dots, l-1, \tag{3.14}$$

$$w^l = \frac{\Delta_{l-1}}{h} + \frac{h}{2} F(\tilde{v}^l). \tag{3.15}$$

(w^l is an approximation of $\dot{v}_{NM}(lh)$.) Gragg [11] showed that if the solution of (3.11) is sufficiently smooth in time, then

$$\tilde{v}^n = v_{NM}(nh) + h^2 c_2 + h^4 c_4 + 0(h^6), \qquad n = 1, \dots, l,$$

where c_2 and c_4 are constants. So Stoermer's rule is second-order accurate in time and two steps of active Richardson extrapolation can be applied at each step to obtain a sixth-order accurate approximation of $v_{NM}(nh)$, $n = 1, \dots, P$, in time.

To complete our discussion of the pseudospectral method, we first derive a sufficient condition for the stability of the pseudospectral method and then discuss the rate of convergence of the method.

We only need to determine conditions under which method (3.13)-(3.15) is stable because the stable variant of Stoermer's rule combined with two steps of Richardson extrapolation is also stable under the stability condition derived for method (3.13)-(3.15).

Equation (3.14) can be rewritten as

$$\tilde{v}^{k+1} - 2\tilde{v}^k + \tilde{v}^{k-1} - h^2 F(\tilde{v}^k) = 0, \qquad k = 1, \dots, l-1. \tag{3.16}$$

The potential $q \in L^\infty(\mathbb{R}^3)$, so we can set $V_{NM}(t) \equiv 0$ in (3.11) to determine a sufficient condition for stability. If $V_{NM}(t) \equiv 0$, then (3.16) becomes

$$J_N(\tilde{B}^{k+1} - 2\tilde{B}^k + \tilde{B}^{k-1} + h^2(\Lambda_N \tilde{B}^k + \tilde{B}^k \Sigma_M))S_M = 0, \tag{3.17}$$

where B^k is the $N \times M$ matrix satisfying $\tilde{v}^k = J_N \tilde{B}^k S_M$.

S_M is invertible and J_N is invertible by assumption, so (3.17) implies that

$$\tilde{B}^{k+1} - 2\tilde{B}^k + \tilde{B}^{k-1} + h^2(\Lambda_N \tilde{B}^k + \tilde{B}^k \Sigma_M) = 0. \tag{3.18}$$

Equation (3.18) constitutes a set of NM decoupled equations for the entries $\tilde{B}_{ij}$ given by

$$(\tilde{B}_{ij})^{k+1} - 2(\tilde{B}_{ij})^k + (\tilde{B}_{ij})^{k-1} + h^2\left(\left(\frac{\lambda i}{a}\right)^2 + \left(\frac{j\pi}{2a}\right)^2\right)(\tilde{B}_{ij})^k = 0,$$

$$i = 1, \dots, N, \qquad j = 1, \dots, M.$$

A straightforward calculation shows that $|\tilde{B}ij|\leqq 1$ for all i and j if and only if

$$h^2\left[\left(\frac{\lambda_N}{a}\right)^2+\left(\frac{M\pi}{2a}\right)^2\right]\leqq 4.$$

This inequality leads to the *stability condition*

$$h\leqq\frac{2a}{\sqrt{(\lambda_N)^2+((M\pi)/2)^2}}. \tag{3.19}$$

We note that the right side of (3.19) is easily estimated. Use the well-known asymptotic result $\lambda_N\approx(N-\frac{1}{4})\pi$ as $N\to\infty$ to get the *estimated stability condition*

$$h\leqq\frac{2a}{\pi\sqrt{(N-\frac{1}{4})^2+(M/2)^2}}. \tag{3.20}$$

We now present a result pertaining to the rate of convergence of the proposed pseudospectral method. In the theorem presented below, we prove that the approximate functions $u_{N,M}(\rho, z, t)$ defined by (3.8) converge to a C^∞ solution of problem (3.4)-(3.7) with infinite-order accuracy provided that the coefficients $b_{ij}(t)$ are the exact Fourier-Bessel coefficients. We believe that the solution of the semi-discrete pseudospectral collocation procedure defined by (3.11) and (3.12) also converges to a C^∞ solution of problem (3.4)-(3.7) with infinite-order accuracy; unfortunately, we have been unable to prove this. However, for the calculations that we perform in § 6, this result is not very important. In most of these calculations we consider potentials that are piecewise smooth. In such cases, the solution of problem (3.4)-(3.7) is not C^∞.

THEOREM. *Let $f(\rho, z)\in C^\infty([0,1],[0,2\pi])$ be a function that satisfies*

(i) $f(-\rho, z)=f(\rho, z)$,

(ii) $\partial f/\partial\rho(0, z)=0$,

(iii) $\partial^j f(1, z)/\partial\rho^j=0$ *for* $j=0, 1, 2, \dots,$ *and*

(iv) $(\partial^j f/\partial z^j)(\rho, 0)=0=(\partial^j f/\partial z^j)(\rho, 2\pi)$ *for* $j=0, 1, 2, \dots$.

Then the approximation

$$a_{N,M}(\rho, z)=\sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}J_0(\lambda_i\rho)\sin(\pi jz)$$

with

$$c_{ij}=\frac{4}{\pi J_0'(\lambda_i)^2}\int_0^\pi\int_0^1 f(\rho,\theta)J_0(\lambda_i\rho)\sin(j\theta)\rho\,d\rho\,d\theta$$

converges to $f(\rho, z)$ in the L_2 norm (with weight ρ) with infinite order of accuracy.

Proof. Mimic the argument in Gottlieb and Orszag [10, pp. 29-33] with respect to ρ and then z. The stated conditions permit unlimited integration by parts for c_{ij} in each variable, so the conclusion is justified.

4. Choice of initial data. To implement our algorithm for computing the SFs of a potential $q(x)$ we need to specify initial data (3.5). According to the mathematical theory in [13], the asymptotic expansion (1.5) is valid for any initial data with compact support and finite energy. So, in principle, any reasonable compactly supported initial data can be used in our algorithm to approximate SFs. In our computations we made special choices for the initial data motivated by some simple theoretical considerations.

We first consider the case when the potential $q(x)$ is radially symmetric, i.e., $q(x)=q(r)=q(|x|_2)$. The basic theoretical results about SFs presented in the introduction show that the SFs of a radial potential depend on the order l of the spherical harmonic $Y_l^m(\theta,\varphi)$, but not on the index m. Furthermore, for each value of l, each SF $\sigma_{l,j}$ corresponds to a scattering eigenfunction with the form $h_{l,j}(r,\theta,\varphi)=h_{l,j}(r)Y_l^0(\theta,\varphi)=h_{l,j}(r)S_l(\varphi)$. These results suggest that we should use initial data with the form

$$u(x,0)=h_1(r)S_l(\varphi),\quad u_t(x,0)=h_2(r)S_l(\varphi),\quad l=0,\dots, \tag{4.1}$$

where $h_1(r)$ and $h_2(r)$ are smooth, compactly supported functions, in order to compute the SFs $\sigma_{l,j}$, $j=0,1,\dots$, for each value of l. In the computations reported in § 6, we used cylindrically symmetric initial data with the form (4.1) with the specific values

$$\begin{aligned} u(\rho,z,0)&=\begin{cases}10^5r^2(0.3-r)S_0(\varphi)=10^5r^2(0.3-r) & \text{for } r=\sqrt{\rho^2+z^2}\leqq 0.3,\\ 0 & \text{for } r\geqq 0.3,\end{cases}\\ u_t(\rho,z,0)&=\begin{cases}r(0.3-r)S_0(\varphi)=r(0.3-r) & \text{for } r\leqq 0.3,\\ 0 & \text{for } r\geqq 0.3,\end{cases}\end{aligned} \tag{4.2}$$

to approximate a finite number of leading radial SFs $\sigma_{0,j}$, $j=1,2,\dots$,

$$\begin{aligned} u(\rho,z,0)&=\begin{cases}10^5r^2(0.3-r)S_1(\varphi)=10^5r(0.3-r)z & \text{for } r=\sqrt{\rho^2+z^2}\leqq 0.3,\\ 0 & \text{for } r\geqq 0.3,\end{cases}\\ u_t(\rho,z,0)&=\begin{cases}r^2(0.3-r)S_1(\varphi)=r(0.3-r)z & \text{for } r\leqq 0.3,\\ 0 & \text{for } r\geqq 0.3,\end{cases}\end{aligned} \tag{4.3}$$

to approximate a finite number of leading SFs $\sigma_{1,j}$, $j=1,2,\dots$, and

$$\begin{aligned} u(\rho,z,0)&=\begin{cases}10^5r^2(0.3-r)S_2(\varphi)=10^5(0.3-r)(3z^2-r^2) & \text{for } r=\sqrt{\rho^2+z^2}\leqq 0.3,\\ 0 & \text{for } r\geqq 0.3,\end{cases}\\ u_t(\rho,z,0)&=\begin{cases}r^2(0.3-r)S_2(\varphi)=(0.3-r)(3z^2-r^2) & \text{for } r\leqq 0.3,\\ 0 & \text{for } r\geqq 0.3,\end{cases}\end{aligned} \tag{4.4}$$

to approximate a finite number of leading SFs $\sigma_{2,j}$, $j=1,2,\dots$.

In the second case where the potential is not radially symmetric, there is no decomposition of the SFs according to the spherical harmonics $S_l(\phi)$. However, we have found that if we use initial data with the form (4.1), then we obtain interesting results about the SFs (see § 6). This choice for the initial data is just one out of numerous choices that can be made to compute SFs of nonradial potentials with our algorithm.

5. The computation of the scattering frequencies from the numerical solution. We have thoroughly discussed the important and delicate problem of computing scattering frequencies from a time series in [17], [26], [27], and [28]. Consequently, we only show how to formulate our problem so that we can use the numerical procedure discussed in these papers.

First, assume that the exact solution $u(x,t)$ of problem (1.3), (1.4) is known. By the discussion in § 2, for each fixed $x\in\mathbb{R}^3$,

$$u(x,t)=\sum_{j=1}^{N}c_jp_j(x)\,e^{i\sigma_jt}+\varepsilon_N(x,t)\quad\text{for } t\geqq t_\infty(x), \tag{5.1}$$

where $\varepsilon_N(x,t)=0(e^{\operatorname{Im}\sigma_Nt})\to 0$ as $N\to\infty$.

Fix the spatial location x and set $t=t_n=t_\infty(x)+n\Delta T$, $u^n=u(x,t_n)$, $d_j=c_jp_j(x)\exp(i\sigma_jt_\infty(x))$, $z_j=e^{i\sigma_j\Delta T}$, and $\varepsilon_N^n=\varepsilon_N(x,t_n)$. Then (5.1) becomes

$$u^n=\sum_{j=1}^N d_j(z_j)^n+\varepsilon_N^n, \qquad n=0,1,\dots, \tag{5.2}$$

where $\varepsilon_N^n\to 0$ as $n\to\infty$ with N fixed or as $N\to\infty$ with n fixed. (The positive constant ΔT is called the Prony sample rate.)

Now let v^n be the numerical solution at location x for time t_n. If v^n is computed by a stable and consistent numerical method, then

$$u^n=v^n-E^n, \qquad n=0,\dots,$$

for some error E^n and (5.2) becomes

$$v^n=\sum_{j=1}^N d_j(z_j)^n+\varepsilon_N^n+E^n, \qquad n=0,\dots. \tag{5.3}$$

Equation (5.3) shows that the numerical solution of problem (1.3), (1.4) contains a finite number of poles z_j, plus an error term $(\varepsilon_N^n+E^n)$. If the error is sufficiently small, then v^n contains only a finite number of poles, which we can compute by applying the version of Prony's method presented in [17] until a subset of poles repeats in successive trials.

6. Reliability of the computational method. In this section we present some computational results about the SFs of bounded, cylindrically symmetric potentials with compact support in $\mathbb{R}^3$. We consider a piecewise constant radially symmetric potential for which we can explicitly determine the SFs. We use this example to document the performance and reliability of our algorithm.

The primary check of the reliability of our numerical algorithm is to look for repetitions of the scattering frequencies as various parameters are changed. In particular, assuming that the numerical solution is sufficiently accurate, we change (i) the presumed number p of poles and (ii) the sample locations. By a sample location, we mean the fixed spatial point x, the time series sample rate ΔT, and the starting time $t_\infty(x)$. A calculation is considered accurate insofar as its repeats under a variety of choices (i) and (ii). In general, we have found that SFs start to repeat in successive trials only when the smallest singular value of the Prony data matrices (defined in [17]) has the same order as the numerical error in solving the wave equation. Furthermore, fewer than p SFs repeat when we assume that the solution contains p poles. See [17] for details.

In the first example, we consider the radially symmetric potential defined by

$$q(r)=6 \quad \text{for } 0\leqq r\leqq 0.3, \qquad q(r)=0 \quad \text{for } r>0.3. \tag{6.1}$$

The results in [19], quoted in the introduction to this paper, show that the SFs of $q(r)$ form a doubly indexed sequence $\sigma_{l,j}$, $l=0,1,\dots,j=1,2,\dots$. It is easily established [26] that each SF $\sigma_{l,j}$ is a root of the nonlinear equation

$$\frac{I_{l+1/2}(0.3\beta)}{I'_{l+1/2}(0.3\beta)}=\frac{\beta}{\lambda}\times\frac{K_{l+1/2}(0.3\lambda)}{K'_{l+1/2}(0.3\lambda)}, \tag{6.2}$$

where $'$ denotes the derivative, $\lambda=i\sigma_{l,j}$, $\beta=\sqrt{\lambda^2+6}$, and $I_{l+1/2}$, $K_{l+1/2}$ are the modified Bessel functions defined by

$$K_{l+1/2}(z)=z^{l+1/2}\left(\frac{1}{z}\frac{d}{dz}\right)^l\left(\frac{e^{-z}}{z}\right)$$

and

$$I_{l+1/2}(z) = z^{l+1/2}\left(\frac{1}{z}\frac{d}{dz}\right)^l\left(\frac{\sinh z}{z}\right).$$

In Table 6.1 we list the actual leading SFs ordered by $\operatorname{Im}\{\sigma_{l,j}\} \leqq \operatorname{Im}\{\sigma_{l,j+1}\}$ for each value of l. (These results were obtained by using Muller's method [3] with error tolerances 10^{-10}.)

We approximate the SFs of (6.1) by using the pseudospectral method combined with Prony's method. We use initial data (4.2)-(4.4) to compute approximations of $\sigma_{l,j}$, $l = 0, 1, 2$, respectively. The parameters for the pseudospectral method are $h = 0.01$, $N = 39$, $M = 79$, $a = 1.426$ for $l = 0$; $a = 1.433$ for $l = 1$; and $a = 1.434$ for $l = 2$. The Prony sample rate is $\Delta T = 0.02$ and the sampling location is $(\rho, z) = (\rho_1, z_{44})$, where ρ_1 and z_{44} are collocation points defined by (3.9). For each value of l, the SFs repeated in successive trials in the Prony algorithm when we assumed that the numerical solution at the sampling location contained 28, 29, and 30 SFs. The SFs repeated in successive trials with at least four significant digits, so we have reported all results with four digits. (All computations in this paper were done in single precision on the Cray 2 at the University of Minnesota and the Cray X/MP at the NSF Supercomputer Center in Pittsburgh.) The numerical approximations of the SFs of potential (6.1) are listed in Table 6.2.

Table 6.2 shows that for $l = 0$, our algorithm with the pseudospectral method computes six pairs of leading SFs with two to three significant digits of accuracy; for $l = 1$, the algorithm computes the leading SF with four significant digits of accuracy and the next four pairs of leading SFs with two to three significant digits of accuracy; and for $l = 2$, the algorithm computes the leading pair of SFs with one significant digit of accuracy and the next five pairs of leading SFs with one to two significant digits of accuracy.

TABLE 6.1
Exact scattering frequencies.

$\sigma_{0,j}$	$\sigma_{1,j}$	$\sigma_{2,j}$
$\pm 8.273 + 7.100i$	$8.580i$	$\pm 4.923 + 10.83i$
$\pm 19.55 + 9.521i$	$\pm 13.19 + 8.701i$	$\pm 17.69 + 9.899i$
$\pm 30.35 + 10.87i$	$\pm 24.53 + 10.33i$	$\pm 29.23 + 11.04i$
$\pm 41.02 + 11.82i$	$\pm 35.39 + 11.42i$	$\pm 40.20 + 11.92i$
$\pm 51.62 + 12.55i$	$\pm 46.09 + 12.23i$	$\pm 50.97 + 12.63i$
$\pm 62.18 + 13.15i$	$\pm 56.71 + 12.89i$	$\pm 61.65 + 13.21i$

TABLE 6.2
SF*s of potential* (6.1) *computed by the spectral method.*

$\sigma_{0,j}$	$\sigma_{1,j}$	$\sigma_{2,j}$
$\pm 8.277 + 7.108i$	$8.580i$	$\pm 4.703 + 10.20i$
$\pm 19.55 + 9.540i$	$\pm 13.20 + 8.724i$	$\pm 17.58 + 9.946i$
$\pm 30.35 + 10.89i$	$\pm 24.55 + 10.26i$	$\pm 29.20 + 11.11i$
$\pm 41.05 + 11.79i$	$\pm 35.68 + 11.41i$	$\pm 40.26 + 12.37i$
$\pm 51.75 + 12.62i$	$\pm 46.04 + 12.59i$	$\pm 50.48 + 12.74i$
$\pm 62.52 + 13.39i$		$\pm 60.31 + 13.09i$

Remark 5. In all calculations in this paper that used the pseudospectral method combined with the Prony algorithm, the parameters for the pseudospectral method, the Prony sample rate, and the sampling location are always the same ones used for potential (6.1). In all cases, the SFs repeated in successive trials in the Prony algorithm with at least four significant digits when we assumed that the numerical solution at the sampling location contained 28, 29, and 30 SFs.

For each potential we carried out a numerical convergence study by increasing the number of basis elements, reducing the time step, and changing the sampling location. When the number of basis functions was increased beyond $N = 39$ and $M = 79$, and the time step reduced below $h = 0.01$, the accuracy of the computed SFs at a given sampling location did not improve. When the sampling location was changed for a computed solution using $N \geqq 39$, $M \geqq 79$, and $h \leqq 0.01$, the SFs changed (in absolute value) by an amount $\leqq 10^{-3}$.

Now we discuss the numerical computation of the SFs for the potentials which depend periodically on time. We present an extension of the theory of SFs in this case (the main reference is [5]) and show that our algorithm for computing SFs applies in this case.

Let $q(x, t)$ be a bounded function that vanishes for $|x|_2 > R$, and is periodic in time with period P, so that $q(x, t+P) = q(x, t)$ for all x, t. Consider the equation

$$u_{tt} - \Delta u + q(x, t)u = 0, \quad x \in \mathbb{R}^3, \quad t > 0, \tag{6.3}$$

with initial conditions (1.4). If the initial data has compact support, then under reasonable conditions, it can be shown that for each fixed x

$$u(x, t) \to \sum_{j=1}^{\infty} c_j p_j(x, t)\, e^{i\sigma_j t} \quad \text{as } t \to \infty. \tag{6.4}$$

(The asymptotic expansion is valid in the local energy norm.) The constants c_j depend on the initial data; the functions $p_j(x, t)$ depend on the potential $q(x, t)$ and the spatial location x and are periodic with period P, so that $p_j(x, t+P) = p_j(x, t)$ (except in the case of multiple σ_j when powers of t may also occur); and the complex constants σ_j, the scattering frequencies, depend on the potential, but are independent of the initial data or spatial location. Since any integer multiple of ν, where $\nu = 2\pi/P$ is the frequency of the potential, can be freely added to a scattering frequency σ to get another one, we normalize the SFs by requiring that $0 \leqq \operatorname{Re}(\sigma) < \nu$.

The following facts about SFs have been proved: (i) There are at most a countable number of SFs σ_j and $\operatorname{Im}\{\sigma_j\} \to +\infty$ as $j \to \infty$ [4]. (ii) Each SF depends continuously on q in the uniform (L^∞) norm [4].

Our numerical algorithm for computing SFs of oscillatory potentials is identical to the one used for stationary potentials, except that some discussion about the Prony sample rate ΔT is needed. We have two practical possibilities:

(1) ΔT is an integer multiple of the period P. In this case $p(x, t_\infty(x) + k\Delta T) = p(x, t_\infty(x))$, and our algorithm for computing SFs is identical to the one used for stationary potentials.

(2) $\Delta T = P/m$ where m is an integer. In this case, Prony's method samples m data points per period, and we are able to see SFs of the form $\sigma = \alpha + i\beta$, $-\alpha + i\beta$, $-\nu + \alpha + i\beta$, $\nu - \alpha + i\beta$, where $\nu = 2\pi/P$. See [28, § 5] for a thorough discussion about this choice of sample rate.

It is well known [14] that the SFs of time-independent compactly supported potentials that are positive satisfy $\operatorname{Im}\{\sigma_j\} > 0$ for all $j = 1, 2, \ldots$, that is, all terms in (6.4) decay in time as t increases. However, it was conjectured in [24] (without proof or any numerical computations) that there exists a positive, oscillatory potential which has a growing mode, that is, Im $\{\sigma_j\} < 0$ for at least one SF σ_j in (6.4). We now present some computational results to corroborate this claim.

Consider the cylindrically symmetric potential

$$(6.5) \qquad q(\rho, z, t) = \begin{cases} 500, & 0.4R(1+0.9\sin(\omega\pi t)) \leqq \rho \quad \text{and} \quad r = \sqrt{\rho^2+z^2} \leqq R = 0.3, \\ 0, & \text{otherwise.} \end{cases}$$

We used the pseudospectral method described in § 3 with Prony sample rate $\Delta T = 0.02$ to compute the leading radial SF $\sigma_{0,1}$ of potential (6.5), excited by initial data (4.2), for a finite number of frequencies satisfying $0 \leqq \omega \leqq 100$. The results are listed in Table 6.3. The computations clearly show that potential (6.5) has a growing mode for several different frequencies. In fact, the numerical results in Table 6.3 suggest much more. They also suggest that the frequencies can be grouped into disjoint intervals with the property that the imaginary part of the leading radial SF changes sign as the frequency crosses the boundary of an interval.

TABLE 6.3
Leading radial SFs *for potential* (6.5) *with different frequencies.*

Frequency	Leading radial SF $\sigma_{0,1}$	Frequency	Leading radial SF $\sigma_{0,1}$
5.0	$16.2+0.59i$	14.5	$21.1-0.40i$
8.5	$16.3+0.75i$	15.0	$20.3+1.60i$
9.0	$14.1-0.7i$	15.5	$19.3+1.07i$
9.5	$14.9-3.0i$	16.0	$18.9+0.5i$
11.5	$18.0-3.58i$	16.5	$18.9+0.27i$
11.7	$18.5-3.47i$	17.5	$18.6+0.27i$
12.5	$19.6-3.1i$	20.0	$17.6+0.51i$
13.0	$20.5-2.61i$	25.0	$17.8+0.53i$
14.0	$20.5-1.13i$	50, 100	$17.8+0.53i$

More computational results are presented in [18]. These results are used to conjecture relationships between a potential and its SFs.

7. Conclusions and applications to other problems. In this paper we have presented a novel time-dependent computational algorithm for computing the scattering frequencies of a bounded compactly supported potential. Important points regarding the choice of $t_\infty(x)$, the initial data, the numerical method for solving the perturbed wave equation (1.3), and various sources of error have been discussed. A number of computations have been performed which demonstrate that our computational algorithm is capable of computing a number of leading pairs of SFs for a cylindrically symmetric potential with one to four (usually two) significant digits of accuracy. In particular, the computations reported here indicate that for radial potentials, our algorithm, combined with the pseudospectral method, computes for each spherical harmonic index l, five to six pairs of leading SFs with one to four significant digits of accuracy. For nonradial cylindrically symmetric potentials and initial data corresponding to each spherical harmonic, our algorithm, combined with the pseudospectral method, computes four to five pairs of leading SFs. These SFs repeat correctly to four significant digits when we alter the sampling location x and repeat the same calculation. Therefore on the basis of the radial potential results, we believe that these approximate SFs are accurate to one to four (probably two) significant digits of accuracy.

We have used our computational algorithm for computing SFs to present several interesting conjectures about the relationship between a potential and its SFs [18].

Our computational algorithm can be used to compute SFs in a number of other important scattering problems. It is known that under reasonable mathematical conditions, an asymptotic expansion with the form (1.5) or (6.4) holds in the following situations:

(i) Acoustic scattering in the exterior of a stationary or periodically vibrating obstacle [5], [13].

(ii) Electromagnetic scattering in the exterior of a stationary or periodically vibrating perfect conductor or dielectric [5], [13].

We have already performed some computations on these other problems, which will be reported in the future.

Acknowledgment. The authors thank David Gottlieb, Peter Lax, Walter Strauss, Greg Forest, and Mac Hyman for their interest in this problem and useful suggestions.

This work was carried out while Musheng Wei was visiting the Institute for Mathematics and Its Applications at the University of Minnesota, Ohio State University, and Michigan State University. He would like to thank the people at these institutions for their support and hospitality.

REFERENCES

[1] C. BAUM, *Emerging technology for transient and broad-band analysis and synthesis of antennas and scatterers*, Proc. IEEE, 64 (1976), pp. 1598-1616.

[2] ———, *The singularity expansion method*, in Transient Electromagnetic Fields, L. B. Felsen, ed., Springer-Verlag, New York, 1976.

[3] S. D. CONTE AND C. DE BOOR, *Elementary Numerical Analysis, An Algorithmic Approach*, 3rd ed., McGraw-Hill, New York, 1980.

[4] J. COOPER, G. PERLA MENZALA, AND W. STRAUSS, *On the scattering frequencies of time-dependent potentials*, Math. Methods Appl. Sci., 8 (1986), pp. 576-584.

[5] J. COOPER AND W. STRAUSS, *Abstract scattering theory for time-periodic systems with applications to electromagnetism*, Indiana Univ. Math. J., 34 (1985), pp. 33-83.

[6] G. DELIC, E. J. JANSE VAN RENSBURG, AND G. WELKE, *A non-self-adjoint general matrix eigenvalue problem*, J. Comput. Phys., 69 (1987), pp. 325-340.

[7] G. DELIC AND G. H. RAWITSCHER, *Sturmian eigenvalue equations with a Chebyshev basis*, J. Comput. Phys., 57 (1985), pp. 188-209.

[8] P. DEUFLHARD, *Recent progress in extrapolation methods for ordinary differential equations*, SIAM Rev., 27 (1985), pp. 505-535.

[9] C. DOLPH AND S. CHO, *On the relationship between the singularity expansion method and the mathematical theory of scattering*, IEEE Trans. Antennas Propagation, AP-28 (1980), pp. 888-896.

[10] D. GOTTLIEB AND S. ORSZAG, *Numerical Analysis of Spectral Methods: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[11] W. B. GRAGG, *On extrapolation algorithms for ordinary initial value problems*, SIAM J. Numer. Anal., 2 (1965), pp. 384-404.

[12] P. HENRICI, *Discrete Variable Methods in Ordinary Differential Equations*, John Wiley and Sons, New York, 1962.

[13] P. LAX AND R. PHILLIPS, *Scattering Theory*, Academic Press, New York, 1967.

[14] ———, *Decaying modes for the wave equation in the exterior of an obstacle*, Comm. Pure Appl. Math., 22 (1969), pp. 737-787.

[15] ———, *The acoustic equation with an indefinite energy form and the Schrödinger equation*, J. Funct. Anal., 1 (1967), pp. 37-83.

[16] ———, *A logarithmic bound on the location of the poles of the scattering matrix*, Arch. Rational Mech. Anal., 40 (1971), pp. 268-280.

[17] G. MAJDA, W. STRAUSS, AND M. WEI, *Computation of exponentials in transient data*, IEEE Trans. Antennas Propagation, 37 (1989), pp. 1284-1290.

[18] G. MAJDA AND M. WEI, *Relationships between a potential and its scattering frequencies*, submitted.

[19] R. G. NEWTON, *Analytic properties of radial wave functions*, J. Math. Phys., 1 (1960), pp. 319-347.

[20] ———, *Scattering Theory of Waves and Particles*, 2nd ed., Springer-Verlag, New York, 1982.

[21] R. PRONY, *Essai experimental et analytique...*, J. l'Ecole Polytech. (Paris), 1 cahier 2, 1795, pp. 24-96.

[22] G. H. RAWITSCHER, *Positive energy Weinberg states for the solution of scattering problems*, Phys. Rev. C, 25 (1982), pp. 2196-2213.

[23] W. REINHARDT, *Complex coordinates in the theory of atomic and molecular structure and dynamics*, Ann. Rev. Phys. Chem., 33 (1982), pp. 223-255.

[24] W. STRAUSS, G. MAJDA, AND M. WEI, *Imaginary poles of radial potentials*, Mathemática Aplicada e Computacional, 6 (1987), pp. 17-24.

[25] R. VOIGT, D. GOTTLIEB, AND M. HUSSAINI, *Spectral Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[26] M. WEI, *Numerical computation of scattering frequencies*, Ph.D. thesis, Division of Applied Mathematics, Brown University, Providence, RI, 1986.

[27] M. WEI AND G. MAJDA, *A new theoretical approach for Prony's method*, J. Linear Algebra Appl., 136 (1990), pp. 119-132.

[28] M. WEI, G. MAJDA, AND W. STRAUSS, *Numerical computation of scattering frequencies for acoustic wave equations*, J. Comp. Phys., 75 (1988), pp. 345-358.

[29] A. LAPIDUS, *Computation of radially symmetric shocked flows*, J. Comput. Phys., 8 (1971), pp. 106-118.

[30] J. STRIKWERDA, *Finite Difference Schemes and Partial Differential Equations*, Wadsworth and Brooks/Cole, Pacific Grove, CA, 1989.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 310-337, March 1993

ANALYSIS OF ITERATIVE METHODS FOR THE STEADY AND UNSTEADY STOKES PROBLEM: APPLICATION TO SPECTRAL ELEMENT DISCRETIZATIONS*

YVON MADAY†§, DAN MEIRON‡, ANTHONY T. PATERA§, AND EINAR M. RØNQUIST§¶

Abstract. A new and detailed analysis of the basic Uzawa algorithm for decoupling of the pressure and the velocity in the steady and unsteady Stokes operator is presented. The paper focuses on the following new aspects: explicit construction of the Uzawa pressure-operator spectrum for a semiperiodic model problem; general relationship of the convergence rate of the Uzawa procedure to classical inf-sup discretization analysis; and application of the method to high-order variational discretization.

Key words. Stokes problem, Uzawa decoupling, iterative methods, inf-sup, spectral elements

AMS(MOS) subject classifications. 65I10, 65M70, 65N12, 65N25

1. Introduction. The Stokes equations describe the motion of incompressible viscous fluid flow at very low Reynolds numbers. However, the need to have efficient Stokes solvers is not only limited to inertia free flows, but is also of great importance when solving numerically the full time-dependent Navier-Stokes equations. For moderate Reynolds numbers the nonlinear convective term is often treated explicitly, while the linear (Stokes) part is treated implicitly. In order for this semi-implicit approach to be attractive, efficient unsteady Stokes solvers are required.

Numerous approaches have been proposed for solving the algebraic system of equations resulting from discretization of the steady and unsteady Stokes equations. One approach is to solve the momentum and continuity equations directly in coupled form (e.g., Yamaguchi, Chang, and Brown [45] and Bathe and Dong [4]). This direct approach is general and robust; however, it can be inefficient and memory intensive for large, three-dimensional problems, in particular, for high-order methods. A second approach is to replace the discrete continuity equations with a Poisson equation for the pressure (e.g., Chorin [15], Temam [41], Glowinski and Pironneau [23], Kleiser and Schumann [27], Kim and Moin [26], and Orszag, Israeli, and Deville [35]). This approach decouples the momentum and continuity except on the domain boundary; however, it may require a rediscretization of the continuous problem, and boundary conditions must be supplied for the pressure.

A third approach, which we study more closely in this paper, is to apply a global nested iterative decoupling procedure for the pressure and the velocity. This scheme is an extension of the classical Uzawa algorithm (see Arrow, Hurwicz, and Uzawa [1], Chorin [15], Temam [41], Glowinski [22], and Girault and Raviart [21] for more basic concepts; see Cahouet and Chabard [13], Maday, Patera, and Rønquist [30], Fischer, Rønquist, Dewey, and Patera [19], Bristeau, Glowinski, and Periaux [12], Maday and Patera [31], and Cahouet and Chabard [14] for more recent advances). The Uzawa

* Received by the editors September 25, 1990; accepted for publication (in revised form) March 19, 1992. This work was supported by the Office of Naval Research and Defense Advanced Research Projects Agency contract N00014-89-J-1610, by Office of Naval Research contract N000114-88-K-0188, by National Science Foundation grant ASC-8806925, by Department of Energy Office of Energy Sciences grant DE-AS03-76ER-72012, Applied Mathematical Sciences (KC-07-01-01), and by Nektonics Research Corporation.

† Université de Paris XII, Laboratoire d'Analyse Numérique de l'Université Pierre et Marie Curie, Paris, France.

‡ California Institute of Technology, Applied Mathematics, 217-50, Pasadena, California 91125.

§ Massachusetts Institute of Technology, Room 3-264, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139.

¶ Present address, Nektonics, 875 Main St., Cambridge, Massachusetts 02139.

approach has several attractive features: It is more efficient in terms of computational complexity and memory requirement than a direct approach; it requires no pressure boundary conditions and no rediscretization of the original problem, and hence the convergence proofs for the original problem directly apply. In essence, by using a block Gaussian elimination procedure, this algorithm decouples the original saddle problem into two positive-semidefinite symmetric forms, one for the pressure and one for the velocity. Thus standard iterative procedures such as preconditioned conjugate gradient iteration and multigrid techniques can readily be applied.

In this paper we give a new and detailed analysis of the basic Uzawa algorithm. The paper focuses on the following new aspects: the explicit construction of the Uzawa pressure-operator spectrum for a particular case; the general relationship of the convergence rate of the Uzawa procedure to classical inf-sup discretization analysis [11], [3], and application of the method to high-order variational discretization. The outline of this paper is as follows. We start in § 2 by reviewing the basic discretization of the steady and unsteady Stokes equations based on the equivalent variational forms. In § 3.1 we review the Uzawa method for the steady Stokes problem, and in § 3.2 we consider the full Fourier case. In § 3.3 we proceed by presenting a new continuous analysis for a semiperiodic model problem. The analytical results regarding the good conditioning of the steady Stokes pressure operator are then verified numerically for optimal high-order spectral element discretizations. In § 3.4 we discuss how these results extend to multidimensional spectral element discretizations, and present examples of steady Stokes problems solved by a nested preconditioned conjugate gradient/multigrid iteration scheme. Last, in § 4 we analyze the Uzawa algorithm in the context of solving the unsteady Stokes equations.

2. The Stokes problems.

2.1. Steady Stokes. In this section we consider the steady Stokes problem in d space dimensions: Find a velocity $\mathbf{u}$ and a pressure p in a domain $\Omega \in \mathcal{R}^d$ such that

$$-\mu\Delta\mathbf{u}+\nabla p=\mathbf{f} \quad \text{in } \Omega, \tag{1}$$

$$-\nabla\cdot\mathbf{u}=0 \quad \text{in } \Omega, \tag{2}$$

subject to homogeneous Dirichlet velocity boundary conditions on the domain boundary $\partial\Omega$,

$$\mathbf{u}=\mathbf{0} \quad \text{on } \partial\Omega. \tag{3}$$

Here $\mathbf{f}$ is the prescribed force and μ is the viscosity. As mentioned in the Introduction, the solution to the Stokes problem (1), (2) is of interest, not only in its own right, but also in that it constitutes the major building block in many Navier–Stokes solvers. In this case, $\mathbf{f}$ can be viewed as an augmented force which includes the explicitly treated nonlinear convective contributions.

The equivalent variational formulation of (1), (2) is: Find $(\mathbf{u}, p)$ in $X\times M$ such that

$$\mu(\nabla\mathbf{u}, \nabla\mathbf{w})-(p, \nabla\cdot\mathbf{w})=(\mathbf{f}, \mathbf{w}) \quad \forall\mathbf{w}\in X, \tag{4}$$

$$-(\nabla\cdot\mathbf{u}, q)=0 \quad \forall q\in M, \tag{5}$$

where the proper spaces for $\mathbf{u}$ and p such that (4), (5) is well posed are [11], [21]

$$X=\mathcal{H}_0^1(\Omega), \tag{6}$$

$$M=\mathcal{L}_0^2(\Omega)=\mathcal{L}^2(\Omega)\cap\left\{\phi\in\mathcal{L}^2(\Omega);\ \int_\Omega \phi\, d\Omega=0\right\}. \tag{7}$$

Here $\mathcal{L}_0^2(\Omega)$ is the space of all functions that are square integrable over Ω with zero average, while $\mathcal{H}_0^1(\Omega)$ is the space of all functions that are square integrable, whose derivatives are also square integrable over Ω, and which satisfy the homogeneous boundary conditions (3).

Here we shall consider numerical approximations to the Stokes problem based on the variational form (4), (5): Find $(\mathbf{u}_h, p_h) \in (X_h, M_h)$ such that

$$\mu((\nabla \mathbf{u}_h, \nabla \mathbf{w}))_h - (p_h, \nabla \cdot \mathbf{w})_h = ((\mathbf{f}, \mathbf{w}))_h \quad \forall \mathbf{w} \in X_h, \tag{8}$$

$$-(\nabla \cdot \mathbf{u}_h, q)_h = 0 \quad \forall q \in M_h, \tag{9}$$

where for each value of the parameter h, $X_h \subset X$ and $M_h \subset M$ are compatible subspaces of X and M (see [11], [3], [21], and [9]) that approach X and M as the discretization parameter h goes to zero. In (8), (9) $(\cdot,\cdot)_h$ and $((\cdot,\cdot))_h$ denote evaluation of the continuous inner product $(\cdot,\cdot)$ by Gauss numerical quadrature (note, however, that the $(\cdot,\cdot)_h$ and $((\cdot,\cdot))_h$ may be different).

Choosing appropriate (compatible) discrete spaces X_h and M_h with associated bases, we arrive at a set of algebraic equations given in matrix form as

$$\mu \underline{A}\underline{u}_i - \underline{D}_i^T \underline{p} = \underline{B}\underline{f}_i, \qquad i = 1, \dots, d, \tag{10}$$

$$-\underline{D}_i \underline{u}_i = 0, \tag{11}$$

where $\underline{A}$ is the discrete Laplace operator, $\underline{B}$ is the mass matrix, $\underline{\mathbf{D}}^T = (\underline{D}_1^T, \dots, \underline{D}_d^T)$ is the discrete gradient operator, and the underscore refers to basis coefficients. In (10), (11), we assume that the homogeneous boundary conditions are imposed by eliminating appropriate rows and columns. Note that in the limit as the discretization parameter $h \Rightarrow 0$, $(X_h, M_h) \Rightarrow (X, M)$, and (10), (11) applies even for the continuous case.

2.2. Unsteady Stokes. The unsteady Stokes equations are given by

$$-\mu \Delta \mathbf{u} + \nabla p + \rho \frac{\partial \mathbf{u}}{\partial t} = \mathbf{f} \quad \text{in } \Omega, \tag{12}$$

$$-\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega, \tag{13}$$

with boundary and initial conditions

$$\mathbf{u} = \mathbf{0} \qquad \text{on } \partial\Omega, \tag{14}$$

$$\mathbf{u}(\mathbf{x}, t = 0) = \mathbf{g}(\mathbf{x}) \quad \text{in } \Omega. \tag{15}$$

Here all variables are defined as in the steady case with t representing time, and ρ the density of the fluid. Although there are physical problems in which the unsteady Stokes equations are relevant, the unsteady problem is primarily of interest with regard to its role in unsteady Navier-Stokes calculations.

We proceed directly to the time discretization of (12), (13) by an implicit Euler backward method (readily extended to Crank-Nicolson)

$$-\mu \Delta \mathbf{u}^{n+1} + \nabla p^{n+1} + \rho \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \mathbf{f}^{n+1}, \tag{16}$$

$$-\nabla \cdot \mathbf{u}^{n+1} = 0, \tag{17}$$

in which $(\mathbf{u}^n, p^n)$ represents an approximation of $(\mathbf{u}(\mathbf{x}, n\Delta t), p(\mathbf{x}, n\Delta t))$, and Δt is the time step.

The spatial discretization of (16), (17) follows the same procedure as for the steady case. At each time step we search for a discrete solution $(\mathbf{u}_h^{n+1}, p_h^{n+1})$ in the finite-dimensional (compatible) subspaces $X_h \subset X$ and $M_h \subset M$, and we arrive at a set of algebraic equations to be solved for the nodal values $\underline{\mathbf{u}}^{n+1} = (\underline{u}_1^{n+1}, \dots, \underline{u}_d^{n+1})$ and $\underline{p}^{n+1}$,

$$\mu \underline{A}\underline{u}_i^{n+1} - \underline{D}_i^T \underline{p}^{n+1} + \rho \underline{B}\left(\frac{\underline{u}_i^{n+1} - \underline{u}_i^n}{\Delta t}\right) = \underline{B}\underline{f}_i^{n+1}, \qquad i = 1, \dots, d, \tag{18}$$

$$-\underline{D}_i \underline{u}_i^{n+1} = 0. \tag{19}$$

We note that for any discretization (18), (19) for which $\underline{A}$ is positive-definite symmetric, (18), (19) is unconditionally stable $(\mathbf{f} = \mathbf{0})$,

$$\|\mathbf{u}_h^{n+1}\|_{0,h} < \|\mathbf{u}_h^n\|_{0,h}, \tag{20}$$

as can be readily demonstrated by multiplying (18) and (19) by $\underline{u}_i^{n+1}$ and $\underline{p}^{n+1}$, respectively. In (20) $\|\cdot\|_{0,h}$ denotes the discrete $\mathscr{L}^2$-norm, $\|v\|_{0,h} = ((v, v))_h^{1/2}$.

3. Steady Stokes solvers.

3.1. The Uzawa algorithm.

The classical Uzawa scheme originates from economic theory as a saddle-point approach to solving constrained optimization problems (see Arrow, Hurwicz, and Uzawa [1]). Following Brezzi [11] and Girault and Raviart [21], the Stokes problem (4), (5) can be formulated as the following equivalent saddle-point problem: Find a pair $(\mathbf{u}, p) \in X \times M$ such that

$$\mathscr{T}(\mathbf{u}, q) \leqq \mathscr{T}(\mathbf{u}, p) \leqq \mathscr{T}(\mathbf{v}, p) \quad \forall \mathbf{v} \in X, \quad \forall q \in M, \tag{21}$$

where the quadratic Lagrangian functional $\mathscr{T}: X \times M \to \mathscr{R}$ is defined by

$$\mathscr{T}(\mathbf{v}, q) = \frac{\mu}{2}(\nabla \mathbf{v}, \nabla \mathbf{v}) - (\mathbf{f}, \mathbf{v}) - (q, \nabla \cdot \mathbf{v}). \tag{22}$$

The constraint in the Stokes problem is the incompressibility condition, while pressure plays the role of the Lagrange multiplier. In the case of finding a numerical approximation to the Stokes problem (1)-(3), the equivalence between the discrete formulation (8), (9) and a finite-dimensional saddle-point problem is now readily seen: Find $(\mathbf{u}_h, p_h) \in X_h \times M_h$ such that

$$\mathscr{T}_h(\mathbf{u}_h, q) \leqq \mathscr{T}_h(\mathbf{u}_h, p_h) \leqq \mathscr{T}_h(\mathbf{v}, p_h) \quad \forall \mathbf{v} \in X_h, \quad \forall q \in M_h, \tag{23}$$

where the quadratic Lagrangian functional $\mathscr{T}_h: X_h \times M_h \to \mathscr{R}$ is defined by

$$\mathscr{T}_h(\mathbf{v}, q) = \frac{\mu}{2}((\nabla \mathbf{v}, \nabla \mathbf{v}))_h - ((\mathbf{f}, \mathbf{v}))_h - (q, \nabla \cdot \mathbf{v})_h. \tag{24}$$

In terms of finding the nodal values $\underline{\mathbf{u}}$ and $\underline{p}$ in (10), (11), the classical Uzawa approach to solving the min-max problem (23) is characterized by the following gradient method [41]:

$$\mu \underline{A}\underline{u}_i^{m+1} = \underline{D}_i^T \underline{p}^m + \underline{B}\underline{f}_i, \qquad i = 1, \dots, d; \tag{25}$$

$$\tilde{\underline{B}}\underline{p}^{m+1} = \tilde{\underline{B}}\underline{p}^m - \alpha \underline{D}_i \underline{\mathbf{u}}_i^{m+1}. \tag{26}$$

Here m is the iteration counter, $\alpha \in \mathscr{R}$ is a positive iteration parameter, and $\tilde{\underline{B}}$ is the mass matrix associated with the bilinear form $(\phi, \psi)_h$ for all $\phi, \psi \in M_h$. In (25) we minimize $\mathscr{T}_h(\mathbf{v}, p_h^m)$ for all $\mathbf{v} \in X_h$, while in (26) we try to maximize $\mathscr{T}_h(\mathbf{u}_h^m, q)$ for all $q \in M_h$. For sufficiently small α, the two-level iteration scheme (25), (26) converges to the solution of (10), (11).

As is the case for many gradient algorithms, the Uzawa procedure in the form (25), (26) converges very slowly, especially for large multidimensional problems. The convergence rate can be improved by considering augmented Lagrangian methods (Fortin and Glowinski [20]), or multigrid schemes (Verfurth [44] and Maître, Musy, and Nigon [34]). However, replacing (25), (26) by conjugate gradient iteration can also accelerate the convergence significantly to give very good results. For details in the finite-element context, we refer to Glowinski [22] and Girault and Raviart [21]. In the following we shall demonstrate, both in terms of continuous analysis and numerical examples, that the latter approach is very attractive in terms of conditioning, computational complexity, and parallelism.

We begin with a decoupling of the original saddle problem (10), (11) into two positive (semi)definite symmetric forms, one for the velocity and one for the pressure. First, for each of the velocity components $\underline{u}_i$ from the momentum equations (10), we formally solve

$$\underline{u}_i = \underline{A}^{-1}\underline{D}_i^T\underline{p} + \underline{A}^{-1}\underline{B}\underline{f}_i, \qquad i = 1, \dots, d. \tag{27}$$

We then insert (27) into the continuity equation (11) to arrive at

$$0 = -\underline{D}_i\underline{u}_i = -\underline{D}_i\underline{A}^{-1}\underline{D}_i^T\underline{p} - \underline{D}_i\underline{A}^{-1}\underline{B}\underline{f}_i. \tag{28}$$

Thus the discrete saddle problem (10), (11) can be replaced with the discretely equivalent statement

$$\underline{A}\underline{u}_i - \underline{D}_i^T\underline{p} = \underline{B}\underline{f}_i, \tag{29}$$

$$\underline{S}\underline{p} = -\underline{D}_i\underline{A}^{-1}\underline{B}\underline{f}_i, \tag{30}$$

where the discrete pressure operator

$$\underline{S} = \underline{D}_i\underline{A}^{-1}\underline{D}_i^T \tag{31}$$

is a positive-semidefinite symmetric matrix. Hence, the saddle problem (10), (11) can be solved by first maximizing $\mathcal{T}_h(\mathbf{u}_h, q)$ for all $q \in M_h$ (see (30)), and then minimizing $\mathcal{T}_h(\mathbf{v}, p_h)$ for all $\mathbf{v} \in X_h$ (see (29)).

We now make several comments regarding the system (29), (30). First, we note that the equation set (29), (30) does not correspond to a rediscretization of the continuous problem, that is, (29), (30) is equivalent to (10), (11). This implies that the theoretical error estimates derived for (8), (9) directly apply (in the case of spectral element discretizations, we refer to Maday and Patera [31] and Maday, Patera, and Rønquist [30]). Second, since the system matrices $\underline{S}$ and $\underline{A}$ are symmetric positive (semi)definite, standard elliptic solvers such as conjugate gradient iteration or multigrid techniques can readily be applied. The system (29), (30) is solved by first solving (30) for the pressure $\underline{p}$ and then solving (29) for the velocity $\underline{u}_i$, $i = 1, \dots, d$ with $\underline{p}$ known. Third, the pressure-operator $\underline{S}$ is completely full due to the embedded inverse $\underline{A}^{-1}$, and thus clearly necessitates an iterative approach.

Heuristically we expect the *continuous* pressure-operator s to be close to the identity operator I and therefore to be well conditioned. To see this, we formally apply the Uzawa decoupling procedure to the continuous equations (1), (2) and neglect boundary conditions

$$s \sim \nabla \cdot (\Delta)^{-1}\nabla \sim I. \tag{32}$$

In the discrete case we do *not* expect $\underline{S}$ to be close to the identity matrix $\underline{I}$, but rather the variational equivalent of the identity operator, the mass matrix $\tilde{\underline{B}}$. Hence, we expect that

$$\tilde{\underline{B}}^{-1}\underline{S} \sim \underline{I}, \tag{33}$$

suggesting that we can invert $\underline{S}$ efficiently by conjugate gradient iteration, using the mass matrix $\tilde{\underline{B}}$ as a preconditioner. Note here the importance of the proper choice of bases and numerical quadratures in order to define a matrix $\tilde{\underline{B}}$ that is easy to invert, that is, in order for $\tilde{\underline{B}}$ to be diagonal.

The preconditioned conjugate gradient iteration (outer iteration) for the system (30) takes the form [24], [21],

$$\begin{gathered}\underline{p}_0; \quad \underline{r}_0 = \underline{D}_i \underline{A}^{-1} \underline{B} \underline{f}_i + \underline{S}\underline{p}_0; \quad \underline{\psi}_0 = \tilde{\underline{B}}^{-1} \underline{r}_0; \quad \underline{\phi}_0 = \underline{\psi}_0; \\ a_m = -\frac{\underline{\psi}_m^T \underline{r}_m}{\underline{\phi}_m^T \underline{S} \underline{\phi}_m}, \quad \underline{p}_{m+1} = \underline{p}_m + a_m \underline{\phi}_m, \quad \underline{r}_{m+1} = \underline{r}_m + a_m \underline{S}\underline{\phi}_m, \\ \underline{\psi}_{m+1} = \tilde{\underline{B}}^{-1} \underline{r}_{m+1}, \quad b_m = \frac{\underline{\psi}_{m+1}^T \underline{r}_{m+1}}{\underline{\psi}_m^T \underline{r}_m}, \quad \underline{\phi}_{m+1} = \underline{r}_{m+1} + b_m \underline{\phi}_m,\end{gathered} \tag{34}$$

where m refers to the iteration number, $\underline{r}_m$ is the residual, $\underline{\phi}_m$ is the search direction, $\tilde{\underline{B}}$ is the preconditioner, $\underline{\psi}_m$ is a vector associated with the preconditioning, and a_m and b_m are scalars.

The inner iteration is associated with the evaluation of the matrix-vector product $\underline{S}\underline{\phi}$ in the outer conjugate gradient iteration. From the definition of $\underline{S}$ in (31) this evaluation can be performed as follows:

$$\underline{y}_i = \underline{D}_i^T \underline{\phi} \qquad i = 1, \ldots, d, \tag{35}$$

$$\underline{A}\underline{z}_i = \underline{y}_i \qquad i = 1, \ldots, d, \tag{36}$$

$$\underline{S}\underline{q} = \underline{D}_i \underline{z}_i. \tag{37}$$

We see that for general discretizations, each matrix-vector product evaluation requires d standard elliptic Laplacian solves in $\mathcal{R}^d$. In order for this approach to be efficient for large multidimensional problems, the discrete Laplace operator $\underline{A}$ must be inverted by a fast solver, such as a good preconditioned conjugate gradient solver. In summary, the pressure is computed from (30) by effecting the *nested* inner/outer iteration procedure (34)-(37).

If the condition number of the matrix $\tilde{\underline{B}}^{-1}\underline{S}$ is order unity, we see that the above algorithm requires only order-d elliptic solves, and hence represents an ideal decoupling of the Stokes problem. We also note that the residual $\underline{r}$ in the outer conjugate gradient iteration (34) is precisely the discrete divergence $-\underline{D}_i \underline{u}_i$. This is a useful result, as it allows for direct control of the discrete divergence when specifying the tolerance for the outer iteration. (The proper choice of tolerances in any nested iterative procedure is an important issue, and will be addressed separately in a future paper.)

We now make some general remarks regarding the relation between the inf-sup condition due to Babuska [3] and Brezzi [11], and the accuracy and efficiency by which the pressure can be computed. The necessary and sufficient condition for well-posedness of the saddle problem (8), (9) can be written as: there exists a real $\beta_h > 0$ such that for all $q \in M_h$, there exists $\mathbf{v} \in X_h$,

$$\beta_h \|q\|_{0,h} \leqq \frac{(q, \nabla \cdot \mathbf{v})_h}{|\mathbf{v}|_{1,h}}, \tag{38}$$

where $\|\cdot\|_{0,h}$ is the discrete $\mathcal{L}^2$-norm associated with the pressure mesh (M_h),

$$\|q\|_{0,h}^2 = (q, q)_h = \underline{q}^T \tilde{\underline{B}} \underline{q}, \tag{39}$$

and $|\cdot|_{1,h}$ is the discrete seminorm associated with the velocity mesh (X_h),

$$|\mathbf{v}|_{1,h}^2 = ((\nabla v_i, \nabla v_i))_h = \underline{v}_i^T \underline{A} \underline{v}_i. \tag{40}$$

In (34) we suggest solving (30) by conjugate gradient iteration, using $\tilde{\underline{B}}$ as a preconditioner. To estimate the efficiency of this approach it is of interest to determine the condition number κ^S of the matrix $\tilde{\underline{B}}^{-1}\underline{S}$. For general discretizations it can be shown that the inf-sup parameter β_h is closely related to the minimum eigenvalue of the pressure-operator $\underline{S}$ (see Appendix A)

$$\beta_h^2 = \lambda_{\min}^S, \tag{41}$$

where $\lambda_{\min}^S$ is the minimum eigenvalue of $\underline{S}$ with respect to the mass matrix $\tilde{\underline{B}}$,

$$\lambda_{\min}^S = \min_{\underline{\phi}} \frac{\underline{\phi}^T \underline{S}\underline{\phi}}{\underline{\phi}^T \tilde{\underline{B}}\underline{\phi}}. \tag{42}$$

It can also be shown (see Appendix A) that the maximum eigenvalue of $\underline{S}$ with respect to $\tilde{\underline{B}}$, $\lambda_{\max}^S$, is of order unity, implying that the condition number κ^S is given as

$$\kappa^S = \frac{\lambda_{\max}^S}{\lambda_{\min}^S} = \frac{C}{\beta_h^2}, \tag{43}$$

where C is a constant of order unity. Thus the number of outer conjugate gradient iterations scales like $1/\beta_h$ [24]. If β_h is of order unity, the outer iteration (34) converges in order-one iterations.

The inf-sup parameter β_h also affects the accuracy by which the pressure can be computed; in fact, it can be shown that the error in the pressure p_h is inversely proportional to β_h, [11], [21], [10],

$$\|p - p_h\|_0 \leqq \frac{C}{\beta_h}\left(\inf_{q_h \in M_h} \|p - q_h\|_0 + \cdots\right), \tag{44}$$

where the dots indicate error terms originating from the velocity and forcing terms. However, the velocity remains unaffected by the inf-sup parameter

$$\|\mathbf{u} - \mathbf{u}_h\|_1 \leqq C\left(\inf_{\mathbf{v}_h \in X_{h,0}} \|\mathbf{u} - \mathbf{v}_h\|_1 + \cdots\right), \tag{45}$$

where

$$X_{h,0} = \{\mathbf{v}_h \in X_h \,|\, (\nabla \cdot \mathbf{v}_h, q_h)_h = 0 \;\forall q_h \in M_h\}, \tag{46}$$

thereby proving that the error in velocity is of the same size as the best fit by discrete divergence-free functions. Even though, in some cases, the presence of weakly spurious modes gives rise to poor approximation by $X_{h,0}$, in many interesting cases we have

$$\inf_{\mathbf{v}_h \in X_h} \|\mathbf{u} - \mathbf{v}_h\|_1 \sim \inf_{\mathbf{v}_h \in X_{h,0}} \|\mathbf{u} - \mathbf{v}_h\|_1 \tag{47}$$

[9], [10], [5], [39], [25].

For reasons of accuracy and efficiency we can now see that it is of great importance that β_h be independent of the mesh parameter h. In most finite-element applications the inf-sup parameter is resolution-independent as long as the discrete spaces are compatible. However, in spectral methods this is not the case, and *weakly* spurious modes [42] are observed. These modes are responsible for an inf-sup parameter β_h that depends on the mesh parameter $h \sim 1/N$, where N is the polynomial degree chosen for the approximation. For example, in the pure spectral case when X_h and M_h consist of all polynomials of degree $\leqq N$, and all (strong) spurious modes for the pressure are eliminated [9], there still exist weakly spurious modes responsible for an inf-sup parameter $\beta_h \sim \mathcal{O}(h) \sim \mathcal{O}(N^{-1})$ [7], [43]. This has led to the construction of alternative methods based on staggered meshes in order to avoid strong spurious modes and to minimize the effects of weakly spurious modes. We refer to Bernardi and Maday [6], and §§ 3.3 and 3.4 for a description of such methods.

The Uzawa algorithm is well known as an efficient way of solving the algebraic system of equations (10), (11) resulting from (low-order) finite-element discretization of the steady Stokes problem (1)-(3). The adaptation of the method to the spectral case is rather new, as described in Maday, Patera, and Rønquist [30], Streett, Hussaini, and Maday [40], Azarez, Labrosse, and Vandeven [2], and Lequéré [29]. In the following we investigate in detail the conditioning of the steady Stokes pressure operator; in particular we shall look at the full Fourier case, a semiperiodic model problem (both continuous and spectral element case), and the multidimensional spectral element case.

3.2. Full Fourier case. We start by first considering the simple case of Fourier discretization in $\mathscr{R}^d$, in which we choose the approximation spaces X_h and M_h to be

$$X_h = M_h^3, \tag{48}$$

$$M_h = \text{span}\,\{e^{i\mathbf{k}\cdot\mathbf{x}}, |k_j| < \mathscr{K}, \forall j = 1, \dots, d\} \tag{49}$$

where $\mathbf{k} = (k_1, k_2, k_3)$ is the wave vector, $\mathbf{x} = (x_1, x_2, x_3) \in \Omega$, and $\mathscr{K}$ is the maximum wave number in each spatial direction. Reality is imposed by conjugate symmetry. For this Fourier discretization it is clear that

$$\underline{B} \Rightarrow 1, \tag{50}$$

$$\underline{D} \Rightarrow ik_j, \tag{51}$$

$$\underline{A} \Rightarrow -k^2 = -\sum_{j=1}^{d} k_j k_j, \tag{52}$$

from which it follows that $\underline{S} = \underline{I}$ independent of $\mathscr{K}$. For the Fourier case the Uzawa algorithm is perfectly conditioned, as might be expected; see Maday and Quarteroni [33] for a numerical analysis of this spatial discretization.

3.3. Semiperiodic case.

3.3.1. Continuous case. Next, we turn to the analysis of the semiperiodic problem. This problem includes boundaries, and is thus much more instructive than the full Fourier case, yet it is sufficiently simple to allow for a complete analysis. The semiperiodic model problem corresponds to the domain $\Omega =]-1, 1[\times]0, 2\pi[$, with (x, y) denoting a point in Ω. The semiperiodic boundary conditions we consider are

$$\forall y \in]0, 2\pi[, \qquad \mathbf{u}(-1, y) = \mathbf{u}(1, y) = \mathbf{0}, \tag{53}$$

$$\forall x \in]-1, 1[, \qquad \mathbf{u}(x, 0) = \mathbf{u}(x, 2\pi), \tag{54}$$

and the associated spaces are

$$X = \{v \in \mathscr{H}^1(\Omega) | v \text{ satisfies (53)-(54)}\}, \tag{55}$$

$$M = \mathscr{L}_0^2(\Omega), \tag{56}$$

where $\mathscr{L}_0^2$ is defined in (7).

We now write the velocity, the pressure, and the data as a Fourier series in the periodic y-direction,

$$\mathbf{u}(x, y) = \sum_{k=-\infty}^{\infty} \hat{\mathbf{u}}^k(x)\, e^{iky}, \tag{57}$$

$$p(x, y) = \sum_{k=-\infty}^{\infty} \hat{p}^k(x)\, e^{iky}, \tag{58}$$

$$\mathbf{f}(x, y) = \sum_{k=-\infty}^{\infty} \hat{\mathbf{f}}^k(x)\, e^{iky}, \tag{59}$$

and use the orthogonality of the Fourier modes to reduce the steady Stokes problem to a series of decoupled (continuous) problems: Find $\hat{\mathbf{u}}^k = (\hat{u}^k, \hat{v}^k)$ and $\hat{p}^k$ in $\hat{X} \times \hat{M}$ such that

$$\hat{u}^k_{xx} - k^2 \hat{u}^k = \hat{p}^k_x + \hat{f}^k, \tag{60}$$

$$\hat{v}^k_{xx} - k^2 \hat{v}^k = ik\hat{p}^k + \hat{g}^k, \tag{61}$$

$$\hat{u}^k_x + ik\hat{v}^k = 0, \tag{62}$$

where

$$\hat{X} = \mathscr{H}^1_0(\Lambda), \tag{63}$$

$$\hat{M} = \mathscr{L}^2_0(\Lambda), \tag{64}$$

$\Lambda =]-1, 1[$, subscript x denotes differentiation with respect to x, $\hat{\mathbf{f}}^k = (\hat{f}^k, \hat{g}^k)$, and we consider wave numbers $k \neq 0$.

From (60)-(62) we now readily derive the following expression for the continuous pressure operator s^k acting on any $\hat{p}^k$ associated with wave number k in the periodic y-direction

$$s^k(\hat{p}^k) = \int_{-1}^{1} \left[\frac{\partial}{\partial x} G(x, x') \frac{\partial}{\partial x'} - k^2 G(x, x') \right] \hat{p}^k(x')\, dx', \tag{65}$$

where $G(x, x')$ is the Green's function for the second-order problem:

$$G_{xx} - k^2 G = \delta(x - x'), \tag{66}$$

$$G(-1, x') = G(1, x') = 0. \tag{67}$$

The solution to (66), (67) can be expressed in closed form as

$$G(x, x') = \frac{-1}{k \sinh 2k} \sinh k(1 + x^<) \sinh k(1 - x^>), \tag{68}$$

where

$$x^< = \begin{cases} x & \text{for } x < x', \\ x' & \text{for } x > x', \end{cases} \tag{69}$$

$$x^> = \begin{cases} x' & \text{for } x < x', \\ x & \text{for } x > x'. \end{cases} \tag{70}$$

To find the condition number of s^k, we analyze the spectrum of the following Fredholm integral equation

$$\int_{-1}^{1} \mathscr{G}(x, x') \chi(x')\, dx' = \lambda^S \chi(x), \tag{71}$$

where the kernel $\mathcal{G}$ follows from (65) as

$$\mathcal{G}(x,x')=\frac{\partial}{\partial x}G(x,x')\frac{\partial}{\partial x'}-k^2G(x,x'). \tag{72}$$

Substituting the expression (68) for the Green's function into (72), the solution to (71) can be found by inspection (see Appendix B). The *entire* spectrum of s^k is given as

$$\lambda_1^s(k)=\frac{1}{2}-\frac{k}{\sinh 2k}, \tag{73}$$

$$\lambda_2^s(k)=\frac{1}{2}+\frac{k}{\sinh 2k}, \tag{74}$$

$$\lambda_l^s(k)=1, \qquad l>2, \tag{75}$$

with only one nonunity eigenvalue for each boundary. The fact that there are only two nonunity λ_l^s's is related to the fact that for the Stokes problem, the pressure and velocity are only coupled at boundaries. (This can also be seen by taking the divergence of the momentum equation, which yields $\Delta p=\nabla\cdot\mathbf{f}$ in Ω, but indeterminacy at the boundaries. The proper boundary conditions are, in fact, $\nabla\cdot\mathbf{u}=0$.)

For a given wave number k, the condition number of s^k is given by

$$\kappa^s(k)=\left(\frac{1}{2}-\frac{k}{\sinh 2k}\right)^{-1}. \tag{76}$$

Since the spectrum (73)-(75) is clustered with only three distinct eigenvalues, the outer conjugate gradient iteration in the pressure solver will converge in three iterations independent of the condition number $\kappa^s(k)$. However, this result is only useful for semiperiodic discretizations; for truly multidimensional problems we must consider the condition number for all admissible k. In particular, if we allow wave numbers in the range $1\leqq k<k_{\max}$, we find that

$$\kappa^s=\left(\frac{1}{2}-\frac{1}{\sinh 2}\right)^{-1}\cong 4.46, \tag{77}$$

which *does not depend on* $k_{\max}$, and hence will not depend on the number of discrete degrees of freedom in the system.

3.3.2. Spectral element discretization. The above continuous analysis suggests that even in the presence of walls, the spectrum of the discrete pressure operator $\underline{S}$ with respect to the mass matrix $\underline{\tilde{B}}$ is clustered near unity, with a condition number that should be largely independent of the discretization parameter h. Here we are primarily interested in spectral element discretizations, corresponding to spaces X_h and M_h consisting of piecewise high-order polynomials [36], [31], [37].

In order to construct the discrete pressure operator $\underline{S}$, the decoupled (continuous) equations (60)-(62) for each Fourier mode k are discretized using spectral elements in the nonperiodic x-direction. The discretization procedure starts by breaking up the domain $\Lambda=]-1,1[$ into K equal elements

$$\bar{\Lambda}=\bigcup_{k=1}^{K}\bar{\Lambda}_k. \tag{78}$$

We then choose the subspaces to be

$$\hat{X}_h=\hat{X}\cap\mathcal{P}_{N,K}(\Lambda), \tag{79}$$

$$\hat{M}_h = \hat{M} \cap \mathscr{P}_{N-2,K}(\Lambda), \tag{80}$$

where

$$\mathscr{P}_{n,K}(\Lambda) = \{\Phi \in \mathscr{L}^2(\Lambda);\ \Phi|_{\Lambda_k} \in \mathscr{P}_n(\Lambda_k)\}, \tag{81}$$

and $\mathscr{P}_n(\Lambda_k)$ denotes the space of all polynomials of degree less than or equal to n with respect to x. The discretization parameter h is thus characterized by two numbers, the number of elements K, and the polynomial degree within each element N. In the following, we shall use the notation $h=(K, N)$. We refer to Maday, Patera, and Rønquist [30] and Bernardi, Maday, and Métivet [10] for a justification of the choice of discrete spaces.

The velocity and pressure are now expressed in terms of high-order Lagrangian interpolant bases through the Gauss-Lobatto and Gauss points, respectively [31]. This choice of bases results in minimal interelemental couplings, while still preserving the required C^0-continuity of the velocity across elemental boundaries. The inner products in (8), (9) are evaluated using Gauss numerical quadrature [16], Gauss-Legendre for $(\cdot,\cdot)_h$, and Gauss-Lobatto-Legendre for $((\cdot,\cdot))_h$. Choosing appropriate test functions, we arrive at a set of algebraic equations of the form (10), (11), which are then decoupled into the form (29), (30). Note that for Legendre spectral element discretizations, the quadrature points are the same as the collocation points, resulting in *diagonal* mass matrices $\underline{B}$ and $\tilde{\underline{B}}$ associated with the staggered mesh. This fact makes the preconditioning in (34) trivial.

We now proceed with the investigation of the conditioning of the discrete pressure operator resulting from spectral element discretization of the semiperiodic model problem. In Maday, Patera, and Rønquist [30], it is shown that the inf-sup parameter $\beta_h(k)$ associated with a particular wave number k is *independent* of the discretization $h=(K, N)$; see also [42] for another proof of this point. As long as the condition number κ^S is of order unity, this result is optimal with regard to both the accuracy of the discrete pressure and the efficiency by which the pressure can be computed. We now present numerical results demonstrating the good conditioning of the preconditioned pressure matrix $\tilde{\underline{B}}^{-1}\underline{S}$ for the semiperiodic problem; in what follows, $\lambda_l^S(k)$, $\kappa^S(k)$ will refer to the spectrum and conditioning of $\tilde{\underline{B}}^{-1}\underline{S}$ for a particular wave number k. The calculation of the eigenvalues is based on EISPACK routines.

We begin by plotting in Fig. 1 the $\lambda_l^S(k)$ for the spectral element discretization $h=(K, N)=(4,7)$ and wave number $k=1$. The agreement with the continuous operator spectrum is seen to be virtually exact. In Fig. 2 we again plot $\lambda_l^S(k)$ with $h=(K, N)=(4,7)$, but now for a wave number $k=12$. The low modes of the system are again in good agreement with the continuous spectrum. However, at this large value of k, the discrete system can no longer resolve exactly the higher modes, resulting in a cluster of eigenvalues at $\lambda_*^S \sim 1.2$. If we investigate the spectrum for $k=12$, but now using a discretization $h=(K, N)=(4,14)$, we see in Fig. 3 that the cluster of numerical eigenvalues has almost disappeared due to the higher spatial resolution in x.

In Fig. 4 we plot κ^S and κ^s as a function of k for the spectral element and continuous operators, respectively. For small and moderate k the two curves coincide; however, as $k \Rightarrow \infty$ the resolution becomes too low and the two curves diverge. For finer resolutions (e.g., larger N) the spectral element and the theoretical results agree over a larger range of wave numbers, as expected from Figs. 2 and 3. For large wave numbers k, the condition number $\kappa^S(k)$ for the spectral element discretization is larger than the value predicted by the continuous analysis, however, the value is still of order unity, as required for fast convergence of the outer iteration.

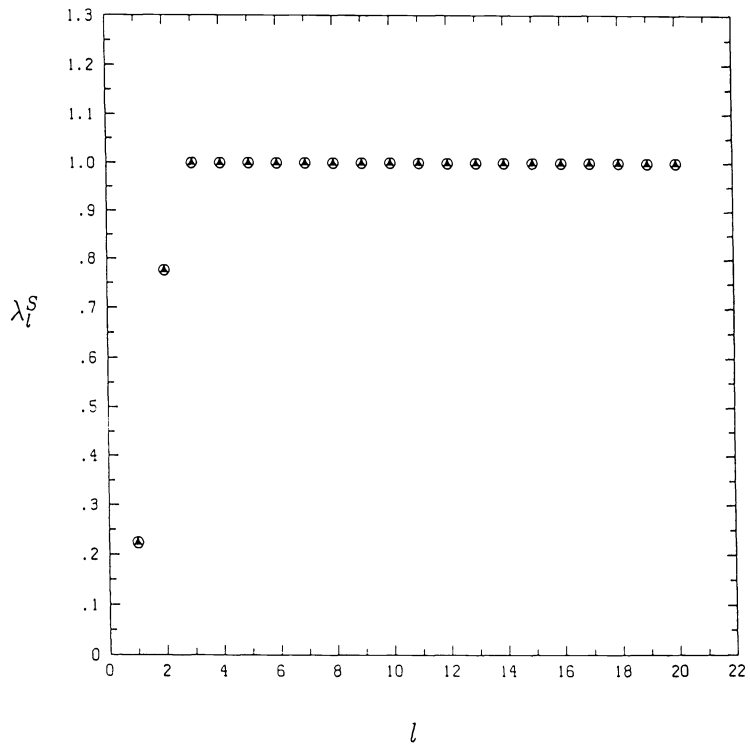


FIG. 1. *A plot of the spectrum $\lambda_l^S(k)$ of the preconditioned steady Stokes pressure matrix $\tilde{\underset{\sim}{B}}^{-1}\underset{\sim}{S}$, where $\underset{\sim}{S}$ is the pressure matrix given in* (31) *and $\tilde{\underset{\sim}{B}}$ is the mass matrix defined on the Gauss pressure mesh. The spectrum* (▲) *corresponds to a spectral element discretization* ($K=4$, $N=7$) *for a wave number $k=1$; the agreement with the continuous operator spectrum λ_l^s of* (73)–(75) (○) *is very good.*

3.4. Multidimensional spectral element case. Before we present any numerical results, we make some general remarks regarding iterative solvers. First, one major reason for using iterative solvers is to avoid the severe memory requirements associated with direct methods, especially for large multidimensional problems. The computational complexity associated with an iterative solver is essentially determined by two factors: the convergence rate of the method, and the operation count for a typical matrix-vector product evaluation. In this section we focus mostly on the conditioning of the Uzawa operator $\underset{\sim}{S}$, which is directly related to the convergence rate of the outer pressure iteration (34). However, we should point out that in the context of high-order methods, fast matrix-vector product evaluations are typically effected by a combination of tensor-product forms and vectorization.

The spectral element discretization procedure for the general multidimensional case is essentially a tensor-product extension of the (one-dimensional) procedure described in § 3.3. In summary, the key points are the use of variational projection operators, piecewise high-order polynomial subspaces, and tensor-product bases and quadratures, resulting in minimal interelemental couplings and efficient matrix-vector product evaluations.

We consider now the Uzawa decoupling procedure (29), (30) as applied to multidimensional spectral element approximations. As discussed earlier, the pressure $\underset{\sim}{p}$ is first computed from (30) by effecting a nested inner/outer iteration procedure, while (29) is solved for the velocity $\underset{\sim}{u}_i$, $i=1,\ldots,d$, with known pressure $\underset{\sim}{p}$. The number

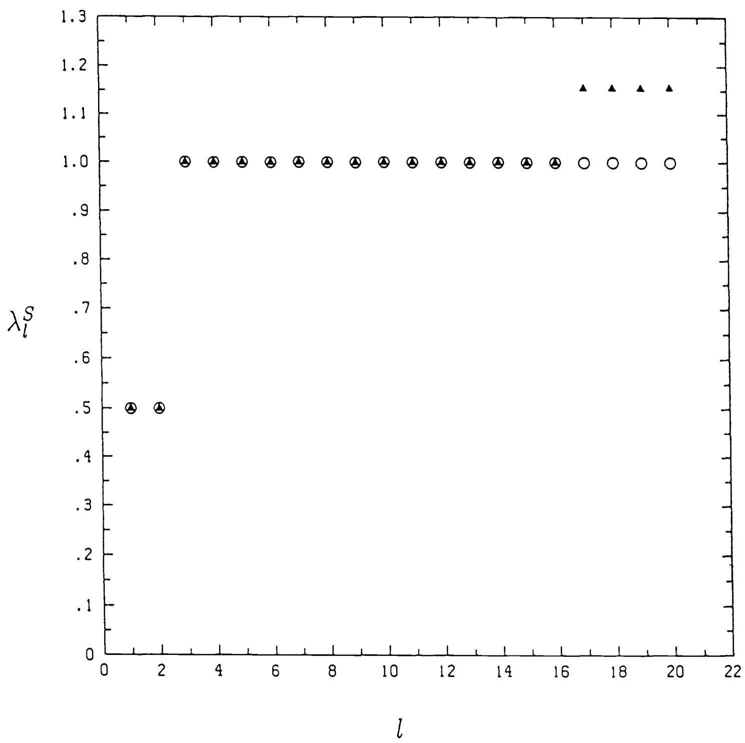


FIG. 2. *A plot of the spectrum* $\lambda_l^S(k)$ *of the preconditioned pressure matrix* $\tilde{\underline{B}}^{-1}\underline{S}$. *The spectrum* (▲) *corresponds to a spectral element discretization* ($K=4$, $N=7$) *for a wave number* $k=12$; *for this large value of* k *the discrete system can no longer resolve the higher continuous modes* (○).

of outer conjugate gradient iterations in (34) critically depends on the condition number κ^S, which we now investigate for multidimensional problems with Dirichlet velocity boundary conditions.

In order to find the condition number κ^S, we must compute the minimum and maximum eigenvalues of the matrix $\underline{S}$ with respect to the mass matrix $\tilde{\underline{B}}$. Since we never form any global system matrix explicitly, standard routines for calculating eigenvalues cannot be used. Instead, we compute the maximum eigenvalue $\lambda_{\max}^S$ using the ordinary power method [24], which involves the evaluation of matrix-vector products of the form $\underline{S}\underline{\phi}$. To compute the minimum eigenvalue $\lambda_{\min}^S$ we use the inverse power method [24], which requires inverting the matrix $\underline{S}$ for each iteration. Note that in order to do this inversion, we use the inner/outer iteration procedure described in (34)-(37).

We start by considering the solution to the steady Stokes equations (1), (2) on a square domain $\Omega =]-1, 1[^2$ with homogeneous Dirichlet velocity boundary conditions (3). Using a spectral element discretization with one single element, i.e., $K=1$, we compute the minimum eigenvalue $\lambda_{\min}^S$ of the discrete steady Stokes pressure-operator $\underline{S}$ for different values of the polynomial degree N. As we can see from Fig. 5, $\lambda_{\min}^S$ decreases as N increases, implying that the β_h in (41) is no longer independent of the discretization $h=(K, N)$. This is, in fact, numerical evidence of the presence of weak spurious modes [42]. Theoretically [8], the results indicate that $\beta_h \sim \mathcal{O}(N^{-1/2})$ as $N \to \infty$, and hence the number of outer conjugate gradient iterations would, at worst, scale like $N^{1/2}$. Note that the numerical results of Fig. 5 show that the theoretical

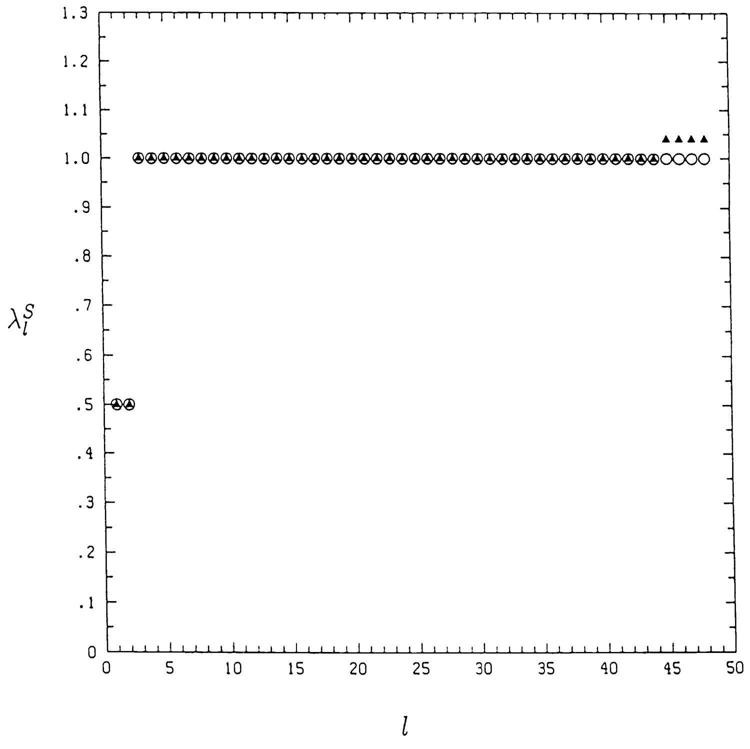


FIG. 3. *A plot of the spectrum* $\lambda_l^S(k)$ *of the preconditioned pressure matrix* $\tilde{\underline{B}}^{-1}\underline{S}$. *The spectrum* (▲) *corresponds to a spectral element discretization* ($K = 4$, $N = 14$) *for a wave number* $k = 12$; *due to the higher spatial resolution the agreement of the discrete spectrum with the continuous operator spectrum* (○) *is improved.*

result is pessimistic for the low values of N of interest in the spectral element context. Next, to see the effect of breaking up the domain Ω into several subdomains, we compare in Fig. 6 the minimum eigenvalue $\lambda_{\min}^S$ when using one single spectral element $K = 1$, and when using $K = 4$ and $K = 16$ equal spectral elements. The results clearly show that β_h is very insensitive to K, especially for larger N.

In Appendix A it is shown that the maximum eigenvalue $\lambda_{\max}^S$ is bounded from above, and that this bound is of order unity. To demonstrate numerically that this is indeed the case, in Fig. 7 we plot $\lambda_{\max}^S$ for different values of the polynomial degree N. The results show that the maximum eigenvalue is insensitive to the number of elements K and to asymptotes to a value below two, as N increases. In practice, the polynomial degree N is typically taken to be of order ten, suggesting that the outer pressure iteration will converge in order-one iterations. Our experience from solving a large variety of two-dimensional and three-dimensional problems is that about ten outer iterations suffice in most cases. We refer to Maday, Patera, and Rønquist [32] for theoretical proofs of the previous numerical evidence.

We now consider a two-dimensional steady Stokes test problem where preconditioned conjugate gradient iteration in the outer pressure iteration is combined with spectral element multigrid for the inner Laplacian solves. The test problem is creeping flow in a "wedge," but with the tip of the wedge removed. The spectral element discretization ($K = 40$, $N = 8$) is shown in Fig. 8(a), and the solution in the form of streamlines is shown in Fig. 8(b). In this test problem, we have removed the tip of the

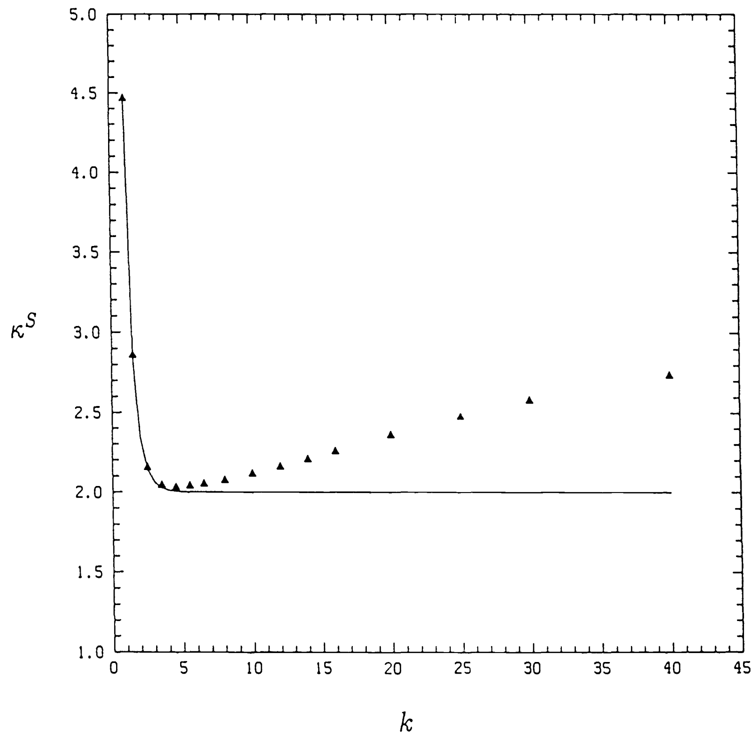


FIG. 4. *A comparison between the condition number $\kappa^S(k)$ of the spectral element operator $\tilde{\underline{B}}^{-1}\underline{S}$ (▲) and the condition number $\kappa^s(k)$ of the continuous operator s in (65) (solid line), as a function of Fourier wave number k. The spectral element discretization ($K = 4$, $N = 10$) is used. For small and moderate k the two curves coincide. However, as $k \Rightarrow \infty$ the two results diverge due to the finite resolution of the spectral element mesh.*

wedge in order to be able to break up the computational domain into spectral elements with aspect ratio approximately equal to unity (see Fig. 8(a)). As discussed in Rønquist [37], the convergence rate of the spectral element multigrid algorithm deteriorates as the aspect ratio of the elements becomes much different from unity. For this particular steady Stokes test problem the total speedup was about 2.5 using multigrid with $J = 4$ meshes instead of preconditioned conjugate gradient iteration for the inner Laplacian solves (timings on a CRAY-XMP). Note that due to the more inefficient vectorization of the matrix-vector products on the coarser meshes ($j = 1$, 2, and 3) compared to the finest mesh ($j = 4$), the computational cost on the coarser meshes ($j \neq J$) cannot be neglected.

Next, we consider the Uzawa decoupling procedure as applied to a three-dimensional steady Stokes problem (1), (2) in a domain Ω defined by $x_1 \in]0, 2\Gamma[$, $x_2 \in]-1, 1[$, $x_3 =]-1, 1[$, where Γ can be interpreted as the aspect ratio of the system. The prescribed force $\mathbf{f}$ is such that the exact solution is given by $\mathbf{u} = (u_1, u_2, u_3) = ((1 - x_2^2)(1 - x_3^2), 0, 0)$ and $p = \sin \pi x_1/\Gamma \cdot \cos \pi x_2 \cdot \cos \pi x_3$. For large three-dimensional problems it is a nontrivial task to compute the eigenvalues of the pressure operator $\underline{S}$, and we therefore instead produce convergence histories from which appropriate condition numbers can be inferred. In particular, we shall plot the residual $\|\underline{r}\|_{0,h}$ (essentially the root mean square of the divergence) as a function of the number of iterations m in the outer conjugate gradient iteration (34).

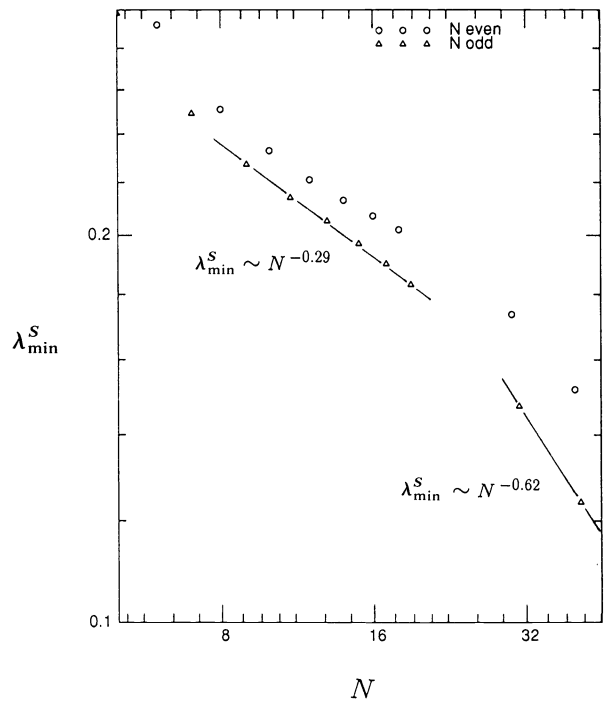


FIG. 5. *A log-log plot of the minimum eigenvalue $\lambda^S_{\min}$ of the discrete Stokes operator $\tilde{B}^{-1}\underline{S}$ as a function of the polynomial degree N (△, N odd; ○, N even). The steady Stokes equations are solved on a square domain $\Omega =]-1, 1[^2$ with homogeneous Dirichlet velocity boundary conditions using $K = 1$ spectral elements. For low values of N, $\lambda^S_{\min} \sim N^{-0.29}$, implying that $\beta_h \sim N^{-0.15}$. For larger values of N, $\lambda^S_{\min} \sim N^{-0.62}$, implying that $\beta_h \sim N^{-0.31}$. Theoretically (Bernardi and Maday [8]), β_h should at worst scale like $N^{-1/2}$, suggesting that the theoretical result is somewhat pessimistic for low values of N.*

In Fig. 9 we plot $\|\underline{r}\|_{0,h}$ as a function of m for an aspect ratio $\Gamma = 1$ and for spectral element discretizations corresponding to $K = 8$, $N = 7$ and 10. The initial convergence rate is almost independent of N, however, the asymptotic convergence rate does appear to be a weak function of N, in good agreement with the above discussion. In Fig. 10 we repeat the numerical experiment of Fig. 9, but now keeping the discretization parameter $h = (K, N) = (8, 10)$ fixed while varying the aspect ratio Γ. The convergence rate is somewhat lower for $\Gamma = 3$ as compared to $\Gamma = 1$, however, the effect is small. These results demonstrate that the good conditioning of the quasi-two-dimensional (semiperiodic) model problem does, indeed, extend to multidimensional problems.

To show the potential of the Uzawa algorithm we present results from a large three-dimensional problem with a complicated geometry. The problem we consider is solving the steady Stokes equations (1), (2) in a spiral-grooved bearing with 16 grooves. Although periodicity conditions could have been exploited, the full three-dimensional problem was discretized using 312,000 degrees of freedom. The set of algebraic equations (29), (30) was then solved on a 64-processor Intel Hypercube in about 16 minutes (160 MFLOPS). The convergence history for the outer pressure iteration (34) is plotted in Fig. 11. We see that the discrete divergence is reduced by three orders of magnitude in about 30 outer iterations. Thus we have demonstrated that the Uzawa algorithm works well for large realistic problems, and can successfully be implemented on a parallel computer. For a more detailed discussion of the parallel aspects of the algorithm, we refer to Fischer and Patera [18].

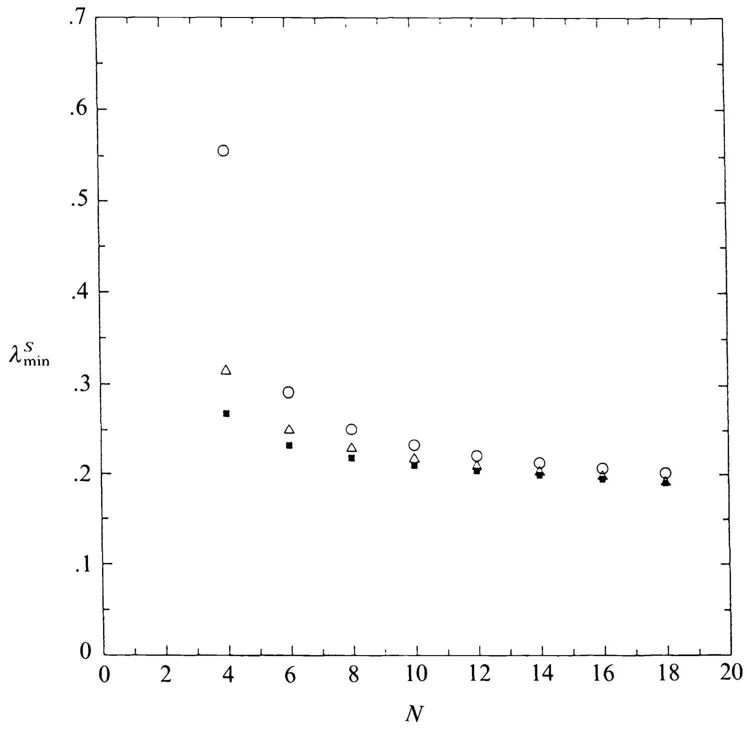


FIG. 6. *A plot of the minimum eigenvalue $\lambda^S_{\min}$ of the discrete Stokes operator $\tilde{\underline{B}}^{-1}\underline{S}$, as a function of the polynomial degree N. The steady Stokes equations are solved on a square domain $\Omega =]-1, 1[^2$ with homogeneous Dirichlet velocity boundary conditions using $K = 1$ (○), $K = 4$ (△), and $K = 16$ (■) spectral elements.*

4. Unsteady Stokes solvers. In § 2.2 we derived a set of algebraic equations (18), (19) resulting from a spectral element discretization of the *implicitly* treated unsteady Stokes problem (12)-(15). In order to compute the nodal values $\underline{u}^{n+1}$ and $\underline{p}^{n+1}$, a classical Uzawa scheme can again be constructed, but now with the discrete Laplace operator $\underline{A}$ in (25) replaced by the discrete Helmholtz operator

$$\underline{H} = \mu\underline{A} + \frac{\rho}{\Delta t}\underline{B}. \tag{82}$$

As in the steady case, the simple gradient method can be accelerated by using conjugate gradient iteration. However, in the unsteady case we must generally consider preconditioners other than the mass matrix $\tilde{\underline{B}}$ [28], [19], [38], [37], [14].

For reasons of efficiency and rigor (no rediscretization), our approach to solving the system (18), (19) will again be based on a global iterative technique. Proceeding in the same fashion as for the steady Stokes case, we arrive at the following decoupled system equivalent to the saddle problem (18), (19)

$$\underline{H}\underline{u}_i^{n+1} = \underline{D}_i^T\underline{p}^{n+1} + \underline{g}_i^n, \qquad i = 1, \dots, d, \tag{83}$$

$$\underline{S}_t\underline{p}^{n+1} = -\underline{D}_i\underline{H}^{-1}\underline{g}_i^n, \tag{84}$$

where

$$\underline{S}_t = \underline{D}_i\underline{H}^{-1}\underline{D}_i^T \tag{85}$$

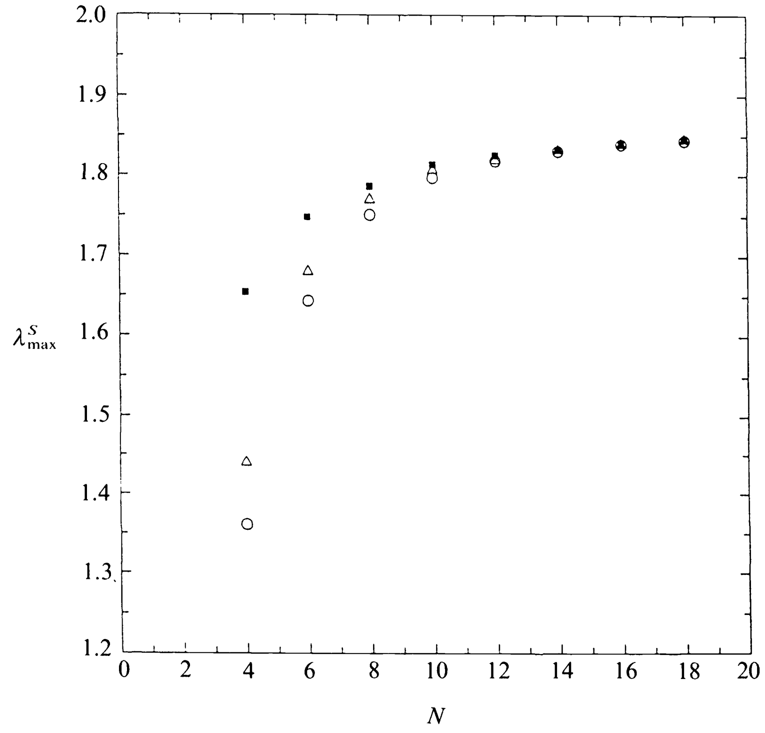


FIG. 7. *A plot of the maximum eigenvalue* $\lambda^S_{\max}$ *of the discrete Stokes operator* $\tilde{B}^{-1}\underset{=}{S}$ *as a function of the polynomial degree N. The steady Stokes equations are solved on a square domain* $\Omega =]-1, 1[^2$ *with homogeneous Dirichlet velocity boundary conditions using* $K = 1$ (○), $K = 4$ (△), *and* $K = 16$ (■) *spectral elements.*

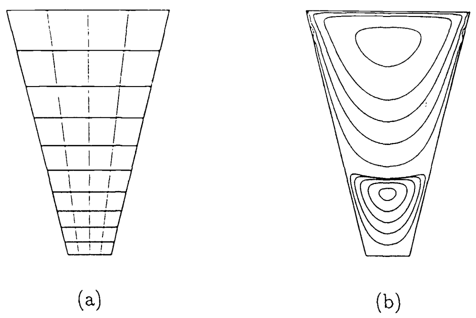


FIG. 8. *Creeping flow in a "wedge" where the tip of the wedge is removed. The imposed velocity boundary conditions are nonslip conditions on the two side walls and at the bottom, with a unit horizontal velocity imposed on the top side*; (a) *shows the spectral element discretization* ($K = 40$, $N = 8$), *while* (b) *shows the solution in form of streamlines.*

is the unsteady Stokes pressure-operator analogous to the steady operator $\underset{=}{S}$ defined in (31), and

$$g_i^n = \underset{=}{B}\left(\underline{f}_i^n + \frac{\rho}{\Delta t}\underline{u}_i^n\right), \qquad i = 1, \ldots, d \tag{86}$$

represent the inhomogeneities associated with an implicit Euler backward time integration procedure. The advantages of the formulation (83), (84) are similar to those for

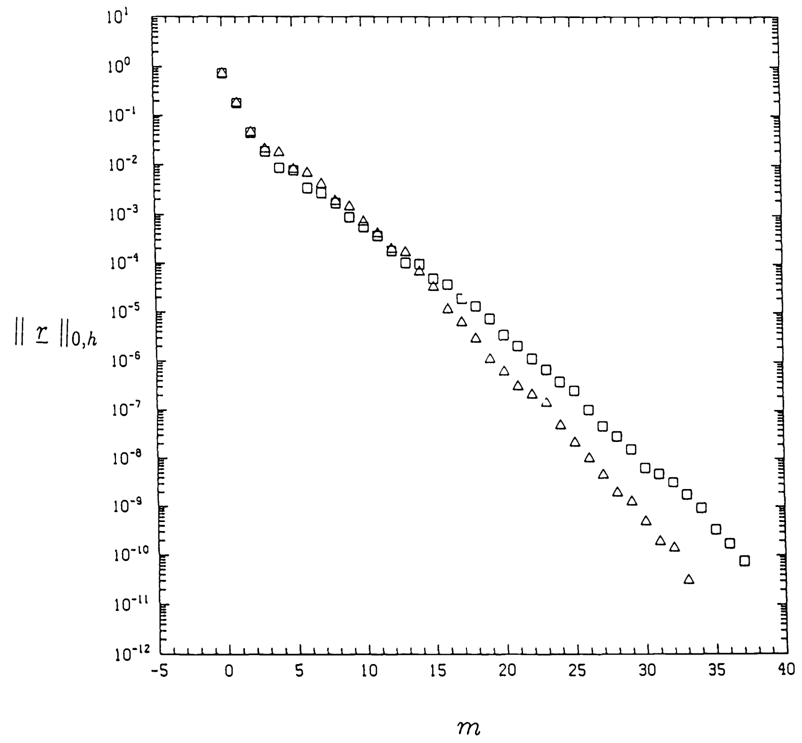


FIG. 9. *A plot of the residual* $\|\underline{r}\|_{0,h}$ *(the root-mean-square of the divergence) from* (34) *as a function of the number of outer conjugate gradient iterations m when solving the three-dimensional steady Stokes problem with solution* $\mathbf{u}=[(1-x_2^2)(1-x_3^2),0,0]$, $p=\sin \pi x_1/\Gamma\cdot\cos \pi x_2\cdot\cos \pi x_3$ *on the domain* $\Omega=]0,2\Gamma[\times]-1,1[\times]-1,1[$ *with* $\Gamma=1$. *The domain is broken up into* $K=8$ *equal spectral elements, with convergence histories shown for* $N=7$ ($\triangle$) *and* $N=10$ ($\square$). *The convergence rate decreases slightly with increasing N.*

the steady problem; it represents a complete, general, velocity-pressure decoupling that is discretely equivalent to the original discretization (18), (19). First, we solve (84) for the pressure, and then (83) is solved for each velocity component $\underline{u}_i^{n+1}$ with $\underline{p}^{n+1}$ known.

As for the steady Stokes problem the matrix S_t is completely full, and therefore solving (84) requires an iterative approach. Unfortunately, whereas the steady pressure-operator $\underline{S}$ is naturally well conditioned ($\underline{\tilde{B}}^{-1}\underline{S}$ is close to the identity), the same is not true for $\underline{S}_t$. For large time steps we can express $\underline{S}_t$ as

$$(87)\qquad \Delta t \Rightarrow \infty, \qquad \underline{S}_t \Rightarrow \frac{1}{\mu}\,\underline{S},$$

and it is thus well conditioned. However, for small time steps, $\underline{S}_t$ goes to the pseudo-Laplacian $\underline{E}$,

$$(88)\qquad \Delta t \Rightarrow 0, \qquad \underline{S}_t \Rightarrow \frac{\Delta t}{\rho}\,\underline{E},$$

where

$$(89)\qquad \underline{E}=\underline{D}_i\underline{B}^{-1}\underline{D}_i^T$$

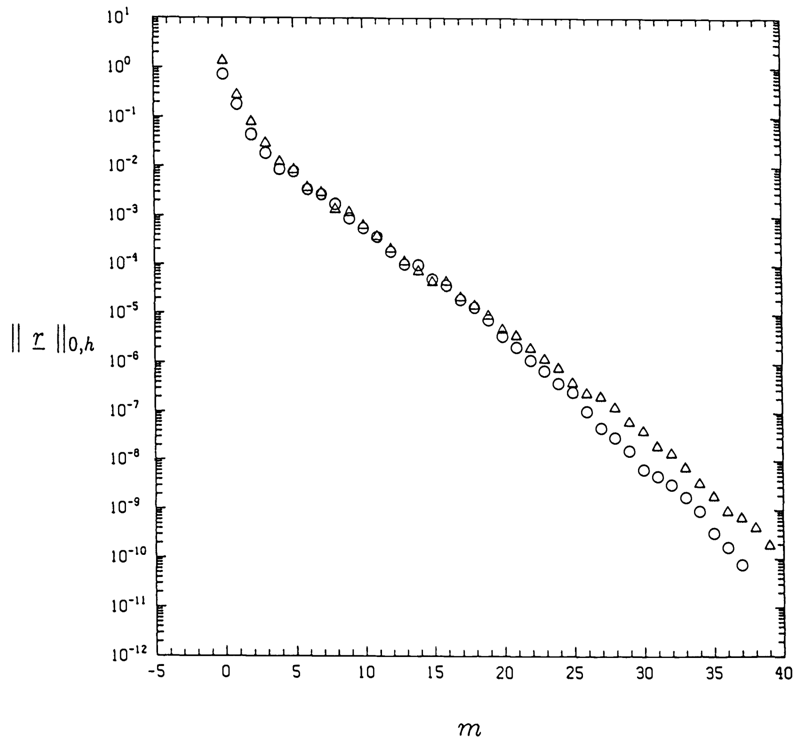


FIG. 10. *A plot of the residual* $\|\underline{r}\|_{0,h}$ *(the root-mean-square of the divergence) from* (34) *as a function of the number of outer conjugate gradient iterations* m *when solving the three-dimensional steady Stokes problem with solution* $\mathbf{u}=[(1-x_2^2)(1-x_3^2),0,0]$, $p=\sin \pi x_1/\Gamma\cdot\sin \pi x_2\cdot\cos \pi x_3$ *on the domain* $\Omega=]-1,\ 1[\times]-1,\ 1[\times]0,\ 2\Gamma[$ *with* $\Gamma=1$ (O) *and* $\Gamma=3$ (△). *Both domains are broken up into* $K=8$ *equal spectral elements, each of order* $N=10$. *The convergence rate decreases slightly as the aspect ratio* Γ *increases.*

is poorly conditioned. The matrix $\underline{E}$ is, in fact, the discrete consistent Poisson-operator resulting from spectral element discretization of the *explicitly* treated unsteady Stokes problem (12)-(15). The algorithm described for the steady case therefore needs to be modified.

Earlier spectral element solvers used a two-level Richardson inner/outer iteration scheme to solve the discrete unsteady Stokes and Navier-Stokes equations [38], [31]. Computational tests indicate that the approach of Cahouet and Chabard [13] is simpler and more efficient, and we shall therefore precondition the unsteady pressure-operator $\underline{S}_t$ directly. The preconditioner proposed is [14]

$$\underline{P}^{-1}=\mu\underline{\tilde{B}}^{-1}+\frac{\rho}{\Delta t}\underline{E}^{-1}, \tag{90}$$

which can be motivated by analyzing the two limits of very small and very large time steps. In both of these cases we expect $\underline{P}^{-1}\underline{S}_t$ to be close to the identity operator. As discussed in Cahouet and Chabard [14], the particular choice (90) as a preconditioner for $\underline{S}_t$ can perhaps be better motivated by considering the Fourier discretization (48), (49) in $\mathcal{R}^d$.

4.1. Multidimensional spectral element case. Our approach to inverting the unsteady pressure-operator is the same as for the steady case, namely, a *nested* global

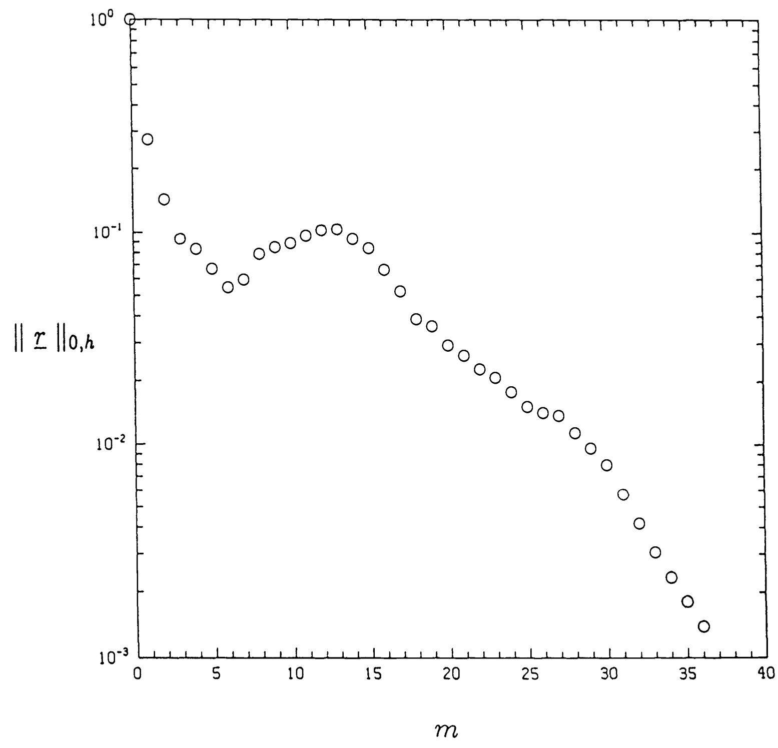


FIG. 11. *A plot of the residual* $\|\underline{r}\|_{0,h}$ (*the root-mean-square of the divergence*) *from* (34) *as a function of the number of outer conjugate gradient iterations m when solving the three-dimensional steady Stokes equations in a spiral-grooved bearing with* 16 *grooves. The* 312,000 *degrees-of-freedom problem was solved on a* 64-*processor Intel hypercube* iPSC/2-VX *in about* 16 *minutes at an average speed of* 160 MFLOPS.

inner/outer iterative procedure based on preconditioned conjugate gradient iteration for the outer iteration, and spectral element multigrid for the inversion of the discrete Helmholtz operator $\underline{H}$. We note that the *structure* in the solution procedure is similar to the steady case, however, the computational *complexity* associated with the preconditioning in the outer iteration is very different. For the steady case the inversion of the *diagonal* mass matrix $\tilde{\underline{B}}$ is trivial, whereas the unsteady case requires the inversion of the pseudo-Laplacian $\underline{E}$. If we count the inversion of the $\underline{E}$-matrix as one standard elliptic solve, each iteration in the outer conjugate gradient iteration takes $d+1$ standard elliptic solves, as compared to d for the steady case. If the condition number of the matrix $\underline{P}^{-1}\underline{S}_t$ is order unity, we see that computing the pressure again requires only order-d elliptic solves. Once the pressure is known, another d elliptic solves is required to compute the velocity.

We now make some remarks regarding the $\underline{E}$-matrix, which is essentially a second-order operator with Neumann-like (pressure) boundary conditions. Our experience from numerical simulations has been that inverting $\underline{E}$ requires more iterations than inverting the standard Laplace operator $\underline{A}$ or Helmholtz operator $\underline{H}$ with Dirichlet (velocity) boundary conditions. The slower convergence rate is probably due to the mixed $\mathscr{L}^2 - \mathscr{H}^1$ spaces in the construction of the $\underline{E}$-matrix. The staggered mesh also makes it more difficult to construct a proper multigrid algorithm. To this end, standard conjugate gradient iteration has been used to invert $\underline{E}$, although a multigrid approach is in preparation.

To demonstrate the effect of the preconditioner $\underset{\sim}{P}$ in (90), we monitor the residual $\|\underset{\sim}{r}\|_{0,h}$ in the outer pressure iteration during the first time step when solving a (simulated) buoyancy-driven flow in a two-dimensional square cavity. We plot in Fig. 12 the convergence history for three different time steps. The larger time step $\Delta t = 1$ is of the order of the time it takes to reach steady state, and $\underset{\sim}{S}_t$ is therefore close to $\underset{\sim}{S}$ $(\mu = 1)$. As expected from the steady Stokes case, we see that about ten outer iterations suffice for convergence. The smaller time step $\Delta t = 10^{-4}$, however, is much smaller than a typical time scale in the system, and $\underset{\sim}{S}_t$ is close to the pseudo-Laplacian $\underset{\sim}{E}$. In fact, the time step is small enough for an explicit time-stepping procedure to be stable, and we see that convergence is reached in order-one iterations. In the limit as the time step $\Delta t \Rightarrow 0$ the unsteady pressure-operator becomes perfectly preconditioned, and the steady Stokes convergence rate represents an upper bound for how fast the outer pressure iteration converges.

For comparison, we repeat in Fig. 13 the experiment of Fig. 12, but now using the preconditioner from the steady Stokes case, that is, $\underset{\sim}{P} = \underset{\sim}{\tilde{B}}$. As expected, as long as $\underset{\sim}{S}_t$ is close to $\underset{\sim}{S}$, the convergence rate is almost identical to the previous case. However, as the time step becomes smaller and $\underset{\sim}{S}_t$ becomes closer to $\underset{\sim}{E}$, the steady Stokes preconditioner does a poor job. In conclusion, the preconditioner (90) is an excellent preconditioner for all time steps.

We close this section by remarking that the Uzawa algorithm can readily be extended to solve the full Navier–Stokes equations by treating the nonlinear convective

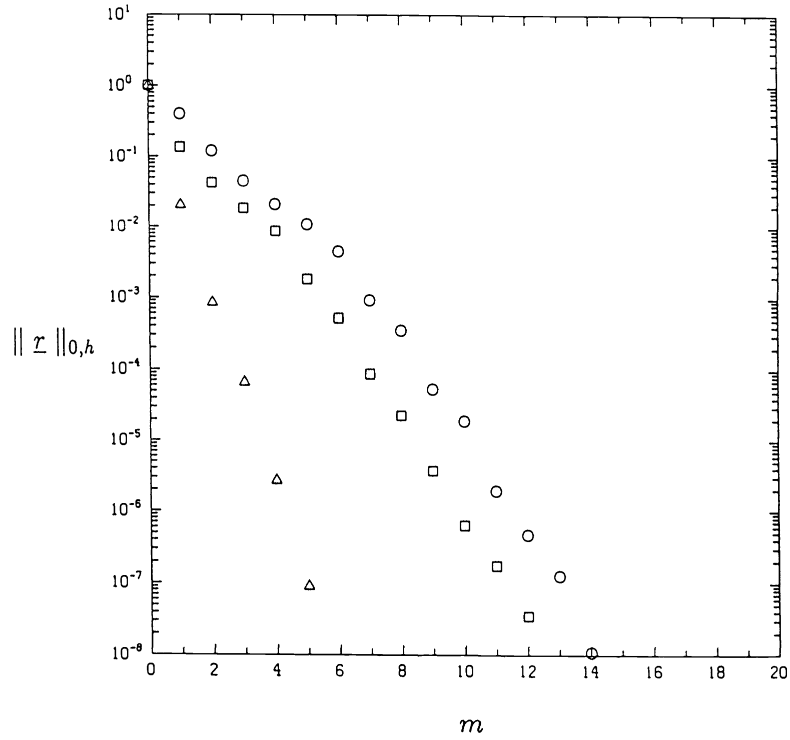


FIG. 12. *A plot of the residual* $\|\underset{\sim}{r}\|_{0,h}$ (*the root-mean-square of the divergence*) *as a function of the number of outer conjugate gradient iterations m when solving for the first time step of a* (*simulated*) *buoyancy-driven flow in a square cavity. The plot shows the convergence history when using* $\underset{\sim}{P}$ *defined in* (90) *as a preconditioner for the unsteady pressure operator* $\underset{\sim}{S}_t$, *for three different time steps* $\Delta t = 10^{-4}$ (△), $\Delta t = 10^{-2}$ (□), *and* $\Delta t = 1$, (○).

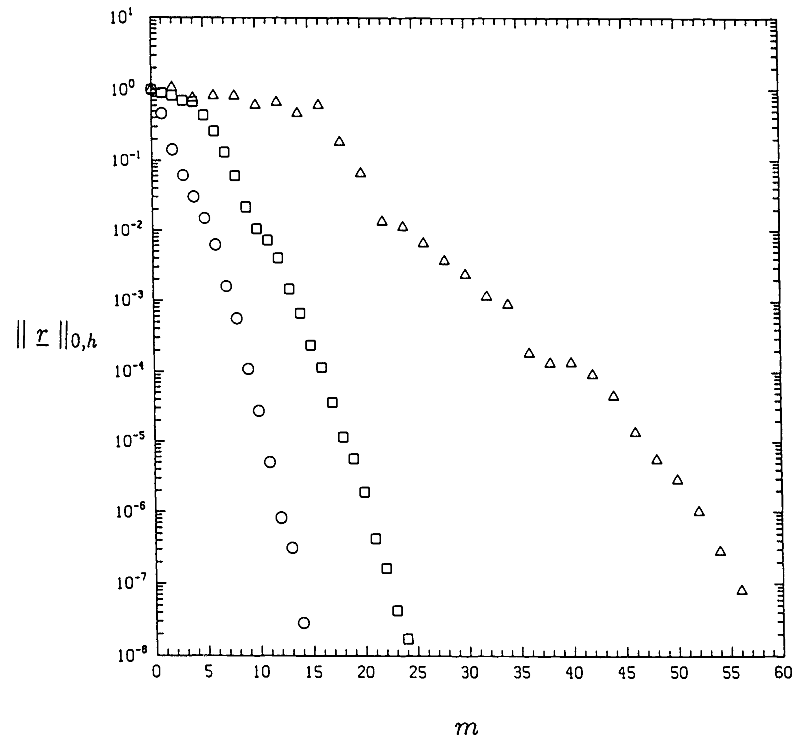


FIG. 13. *A plot of the residual* $\|\underset{\sim}{r}\|_{0,h}$ *(the root-mean-square of the divergence) as a function of the number of outer conjugate gradient iterations m when solving for the first time step of a (simulated) buoyancy-driven flow in a square cavity. The plot shows the convergence history when using* $\tilde{\underset{\sim}{B}}$ *as a preconditioner for the unsteady pressure-operator* $\underset{\sim}{S}_t$ *for three different time steps* $\Delta t = 10^{-4}$ ($\triangle$), $\Delta t = 10^{-2}$ ($\square$), *and* $\Delta t = 1$, ($\bigcirc$).

term explicitly. This approach has been used with success in the context of spectral element discretizations [17].

Appendix A. This appendix deals with relations between the condition number of the algebraic system that arises from the Uzawa algorithm and the various parameters of the discretization, in particular, the inf-sup condition, but also other constants related to the exactness of the integration formulae. To this purpose, let us recall that we have set

$$a_h(u_h, v_h) = ((\nabla u_h, \nabla v_h))_h, \tag{A.1}$$

and introduce the following constants

$$c_1 = \sup_{\phi_h \in M_h} \frac{(\phi_h, \phi_h)_h}{(\phi_h, \phi_h)}, \tag{A.2}$$

$$c_1' = \inf_{\phi_h \in M_h} \frac{(\phi_h, \phi_h)_h}{(\phi_h, \phi_h)}, \tag{A.3}$$

$$\alpha = \inf_{\mathbf{v}_h \in X_h} \frac{a_h(\mathbf{v}_h, \mathbf{v}_h)}{\|\mathbf{v}_h\|_1^2}. \tag{A.4}$$

Let us first bound the maximum eigenvalue $\lambda_{\max}^{S}$ of the matrix $\underset{\sim}{S}$ with respect to the mass matrix $\tilde{\underset{\sim}{B}}$.

To this purpose, let us first consider the discrete Laplace operator defined by $\mathbf{u}_h = \Delta_h^{-1}\mathbf{g}$ if

$$a_h(\mathbf{u}_{h,i}, \mathbf{v}_i) = ((\mathbf{g}_i, \mathbf{v}_i))_h, \quad \forall \mathbf{v} \in X_h. \tag{A.5}$$

Let us now consider the operator div^T, transposed by the divergence operator with respect to the $(.,)_h$-scalar product, i.e.,

$$((\operatorname{div}^T p, \mathbf{v}))_h = (p, \operatorname{div} \mathbf{v})_h, \quad \forall p \in M_h, \quad \forall \mathbf{v} \in X_h.$$

LEMMA A.1. *For any p in $\mathscr{L}^2(\Omega)$, we have*

$$\|\Delta_h^{-1}(\operatorname{div})^T p\|_1 \leqq \frac{c_1}{\alpha}\|p\|_0.$$

Proof. This result is simply derived by taking $\mathbf{v} = \Delta_h^{-1}(\operatorname{div})^T p$ in (A.5) with $g = \operatorname{div}^T p$, and applying the Cauchy-Schwarz inequality.

It is important to note that with these definitions, the operator $\operatorname{div}(\Delta_h^{-1})\operatorname{div}^T$ is symmetric and that $\underset{\sim}{S}$ represents its matrix in the basis of the Lagrange interpolants. Due to the symmetry, the extreme eigenvalues are related to the upper and lower norms of the images of elements of M_h as follows:

$$(\lambda_{\max}^S) = \sup_{p \in M_h} ([\operatorname{div}(\Delta_h^{-1})\operatorname{div}^T]p, p)_h, \qquad (p,p)_h = 1 \tag{A.6}$$

and

$$(\lambda_{\min}^S) = \inf_{p \in M_h} ([\operatorname{div}(\Delta_h^{-1})\operatorname{div}^T]p, p)_h, \qquad (p,p)_h = 1. \tag{A.7}$$

Note that these eigenvalues are also those of the matrix $\underset{\sim}{S}$ with respect to the mass matrix $\underset{\sim}{\tilde{B}}$. From (A.2), (A.3), and Lemma A.1, it is an easy matter to check that

$$\lambda_{\max}^S \leqq \frac{c_1^2}{\alpha c_1'}.$$

Let us now consider $\lambda_{\min}^S$. We want to get the relationship between this eigenvalue and the inf-sup condition constant β_h given by

$$\beta_h = \inf_{p \in M_h} \sup_{\mathbf{v} \in X_h} \frac{(p, \operatorname{div} \mathbf{v})_h}{((\nabla \mathbf{v}_i, \nabla \mathbf{v}_i))_h^{1/2}}, \qquad (p,p)_h = 1.$$

Let p be given in M_h with $(p,p)_h = 1$. As noted by Vandeven [42, V, Théorème II.1], the elements $\mathbf{u}^*$ that realize the supremum related to the inf-sup condition, i.e.,

$$\sup_{\mathbf{v} \in X_h} \frac{(p, \operatorname{div} \mathbf{v})_h}{((\nabla \mathbf{v}_i, \nabla \mathbf{v}_i))_h^{1/2}} = \frac{(p, \operatorname{div} \mathbf{u}^*)_h}{((\nabla \mathbf{u}_i^*, \nabla \mathbf{u}_i^*))_h^{1/2}}, \tag{A.8}$$

are colinear to the element $\tilde{\mathbf{u}}^*$ of X_h, solution of the problem

$$\tilde{\mathbf{u}}^* = \Delta_h^{-1}\operatorname{div}^T p. \tag{A.9}$$

Since the proof of the fact is very short and simple, let us repeat it here. From (A.9) we have

$$\frac{(p, \operatorname{div} \mathbf{v})_h}{((\nabla \mathbf{v}_i, \nabla \mathbf{v}_i))_h^{1/2}} = \frac{((\nabla \tilde{\mathbf{u}}_i^*, \nabla \mathbf{v}_i))_h}{((\nabla \mathbf{v}_i, \nabla \mathbf{v}_i))_h^{1/2}}.$$

The Cauchy-Schwarz inequality then gives the inequality

$$\frac{(p, \operatorname{div} \mathbf{v})_h}{((\nabla \mathbf{v}_i, \nabla \mathbf{v}_i))_h^{1/2}} \leq ((\nabla \tilde{\mathbf{u}}_i^*, \nabla \tilde{\mathbf{u}}_i^*))_h^{1/2},$$

with the equality if and only if $\tilde{\mathbf{u}}_i^*$ and $\mathbf{v}$ are colinear. This leads from (A.9) to

$$\text{(A.10)} \quad \sup_{\mathbf{v} \in X_h} \frac{(p, \operatorname{div} \mathbf{v})_h}{((\nabla \mathbf{v}_i, \nabla \mathbf{v}_i))_h^{1/2}} = (\operatorname{div}^T p, (\Delta_h^{-1}) \operatorname{div}^T p)_h^{1/2} = ([\operatorname{div}(\Delta_h^{-1}) \operatorname{div}^T] p, p)_h^{1/2},$$

so that the inf-sup condition satisfies

$$\text{(A.11)} \qquad \beta_h = \inf_{p \in M_h} ([\operatorname{div}(\Delta_h^{-1}) \operatorname{div}^T] p, p)_h^{1/2}, \qquad (p, p)_h = 1.$$

Now recalling (A.7), we have

$$\lambda_{\min}^S = \beta_h^2.$$

Finally, the condition number κ^S is then bounded by $c_1^2 \beta_h^2 / \alpha c_1'$.

Appendix B. We consider here the solution to the eigenvalue problem (71):

$$\text{(B.1)} \qquad \int_{-1}^{1} \left\{ \frac{\partial}{\partial x} G(x, x') \frac{d\chi(x')}{dx'} - k^2 G(x, x') \chi(x') \right\} dx' = \lambda \chi(x),$$

where the Green's function $G(x, x')$ is given in closed form in (68).

To reduce the integral equation (B.1) to the form of a standard eigenvalue problem, we integrate the term involving the derivative of χ by parts,

$$\text{(B.2)} \qquad \int_{-1}^{1} \frac{\partial G}{\partial x} \frac{d\chi}{dx'} dx' = \frac{\partial G}{\partial x} \chi \Big|_{-1}^{x^-} + \frac{\partial G}{\partial x} \chi \Big|_{x^+}^{1} - \int_{-1}^{1} \frac{\partial^2 G}{\partial x \partial x'} \chi \, dx'.$$

Here the integral is broken up into two parts due to the jump discontinuity in the derivatives of G,

$$\text{(B.3)} \qquad \frac{\partial G}{\partial x'}(x, x^+) - \frac{\partial G}{\partial x'}(x, x^-) = 1.$$

Using (B.2) and (B.3), we can write (B.2) as

$$\text{(B.4)} \qquad \chi(x) - \int_{-1}^{1} \left\{ \frac{\partial^2 G}{\partial x \partial x'} + k^2 G \right\} \chi(x') \, dx' = \lambda \chi(x),$$

which is a homogeneous Fredholm integral equation of the second kind.

Using the explicit form of G from (68) and evaluating the derivatives, we obtain

$$\text{(B.5)} \qquad \begin{aligned} -\int_{-1}^{1} \frac{k^2}{k \sinh 2k} \{ &\cosh k(1 + x^<) \cosh k(1 - x^>) \\ &+ \sinh k(1 + x^<) \sinh k(1 - x^<) \} \chi(x') \, dx' \\ = (\lambda - 1) \chi(x)&. \end{aligned}$$

Using the definitions of $x^<$ and $x^>$ in (69), (70), (B.5) reduces to the following symmetric eigenvalue problem:

$$\text{(B.6)} \qquad \frac{-k}{\sinh 2k} \int_{-1}^{1} \cosh k(x + x') \chi(x') \, dx' = (\lambda - 1) \chi(x).$$

Expanding the kernel in (B.6) we arrive at

$$-\frac{k}{\sinh 2k}\left\{\cosh kx \int_{-1}^{1} \cosh kx'\chi(x')\,dx' + \sinh kx \int_{-1}^{1} \sinh kx'\chi(x')\,dx'\right\} = (\lambda-1)\chi(x). \tag{B.7}$$

By inspection, it is clear that there are two solutions to (B.7) given by

$$\chi^e = \cosh kx, \qquad \lambda^e = \frac{1}{2} - \frac{k}{\sinh 2k}, \tag{B.8}$$

$$\chi^o = \sinh kx, \qquad \lambda^o = \frac{1}{2} + \frac{k}{\sinh 2k}, \tag{B.9}$$

where superscripts e and o denote even and odd, respectively. In addition, there exists an infinite set of eigenfunctions χ corresponding to $\lambda = 1$, satisfying

$$\int_{-1}^{1} \cosh kx'\chi(x')\,dx' = \int_{-1}^{1} \sinh kx'\chi(x')\,dx' = 0. \tag{B.10}$$

For example, we can choose χ odd such that

$$\int_{-1}^{1} \cosh kx'\chi(x')\,dx' = 0, \tag{B.11}$$

or χ even such that

$$\int_{-1}^{1} \sinh kx'\chi(x')\,dx' = 0. \tag{B.12}$$

There are many ways to do this, which explains why the spectrum is clustered around unity.

Acknowledgments. We would like to thank Paul Fischer at Brown University, and Intel Scientific Computers for providing us with the parallel spectral element results.

REFERENCES

[1] K. Arrow, L. Hurwicz, and H. Uzawa, *Studies in Nonlinear Programming*, Stanford University Press, Stanford, CA, 1958.

[2] M. Azarez, G. Labrosse, and H. Vandeven, *A pressure field pseudospectral evaluation for 3D numerical experiments in incompressible fluid dynamics*, in Proc. 11th Internat. Conf. Numerical Methods in Fluid Dynamics, D. L. Dwoyer, M. Y. Hussaini, and R. G. Voigt, eds., Springer-Verlag, Berlin, Heidelberg, New York, 1989.

[3] I. Babuška, *Error bounds for the finite element method*, Numer. Math., 16 (1971), pp. 322-333.

[4] K. J. Bathe and J. Dong, *Solution of incompressible viscous fluid flow with heat transfer*, J. Comput. Structures, to appear.

[5] C. Bernardi, C. Canuto, and Y. Maday, *Generalized inf-sup condition for Chebyshev spectral approximation of the Stokes problem*, SIAM J. Numer. Anal., 25 (1988), pp. 1237-1271.

[6] C. Bernardi and Y. Maday, *A collocation method over staggered grid for the Stokes problem*, Internat. J. Numer. Methods Fluids, 8 (1988), pp. 537-557.

[7] ———, *Relèvement polynomial de traces et applications*, M^2AN, 24 (1990), pp. 557-611.

[8] ———, *Approximation spectrales de problèmes aux limites elliptiques*, Mathématiques et Applications, 10, Ellipse édition marketing, Springer-Verlag, Paris, 1992.

[9] C. Bernardi, Y. Maday, and B. Métivet, *Spectral approximation of the periodic-nonperiodic Navier-Stokes equations*, Numer. Math., 51 (1987), pp. 655-700.

[10] C. BERNARDI, Y. MADAY, AND B. MÉTIVET, *Calcul de la pression dans la résolution spectrale du problème de Stokes*, Recherche Aerospatiale, 1987, pp. 1-21.

[11] F. BREZZI, *On the existence, uniqueness and approximation of saddle-point problems arising from Lagrange multipliers*, RAIRO Anal. Numer., 8 R2 (1974), pp. 129-151.

[12] M. O. BRISTEAU, R. GLOWINSKI, AND J. PERIAUX, *Numerical methods for the Navier-Stokes equations. Applications to the simulation of compressible and incompressible viscous flows*, Comput. Phys. Rep., to appear.

[13] J. CAHOUET AND J. P. CHABARD, *Multi-domains and multi-solvers finite element approach for the Stokes problem*, Proceedings of the Fourth International Symposium on Innovative Numerical Methods in Engineering, R. P. Shaw, ed., Springer-Verlag, New York, 1986, p. 317.

[14] ———, *Some fast 3d finite element solvers for the generalized Stokes problem*, Internat. J. Numer. Methods Fluids, 8 (1988), pp. 869-895.

[15] A. J. CHORIN, *Numerical solution of incompressible flow problems*, in Studies in Numerical Analysis 2, J. M. Ortega and W. C. Rheinboldt, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1970.

[16] P. J. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration*, Academic Press, New York, 1985.

[17] P. FISCHER, L. W. HO, G. E. KARNIADAKIS, E. M. RØNQUIST, AND A. T. PATERA, *Recent advances in parallel spectral element simulation of unsteady incompressible flows*, Comput. & Structures, 30 (1988), pp. 217-231.

[18] P. FISCHER AND A. T. PATERA, *Parallel spectral element solution of the Stokes problem*, J. Comput. Phys., 92 (1991), pp. 380-421.

[19] P. FISCHER, E. RØNQUIST, D. DEWEY, AND A. T. PATERA, *Spectral element methods: Algorithms and architectures*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 173-197.

[20] M. FORTIN AND R. GLOWINSKI, *Augmented Lagrangian Methods*, North-Holland, Amsterdam, 1983.

[21] V. GIRAULT AND P. A. RAVIART, *Finite Element Approximation of the Navier-Stokes Equations*, Springer-Verlag, Berlin, Heidelberg, Germany, 1986.

[22] R. GLOWINSKI, *Numerical Methods for Nonlinear Variational Problems*, Springer-Verlag, Berlin, New York, 1984.

[23] R. GLOWINSKI AND O. PIRONNEAU, *On a mixed finite element approximation of the Stokes problem*, Numer. Math., 33 (1979), pp. 397-424.

[24] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[25] S. JENSEN AND M. VOGELIUS, *Divergence stability in connection with the p-version of the finite element methods*, M^2AN, 24 (1990), pp. 737-764.

[26] J. KIM AND P. MOIN, *Application of a fractional-step method to incompressible Navier-Stokes equations*, J. Comput. Phys., 59 (1985), pp. 308-323.

[27] L. KLEISER AND U. SCHUMANN, *Spectral simulation of the laminar turbulent transition process in plane Poiseuille flow*, in Spectral Methods for Partial Differential Equations, R. G. Voigt, D. Gottlieb, and M. Y. Hussaini, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[28] G. LABADIE AND P. LASBLEIZ, *Quelques métodes de résolution du problème de Stokes en éléments finis*, Tech. Rep. HE41/83.01, Electricité de France, 1983.

[29] P. LEQUÉRÉ, *Mono and multi domain Chebyshev algorithm on a staggered grid*, in Proc. Seventh Internat. Conf. Finite Element Methods in Flow Problems, 1985.

[30] T. MADAY, A. T. PATERA, AND E. M. RØNQUIST, *A well-posed optimal spectral element approximation for the Stokes problem*, Tech. Rep. No. 87-48, 1987, ICASE, Hampton, VA.

[31] Y. MADAY AND A. T. PATERA, *Spectral element methods for the Navier-Stokes equations*, in State-of-the-Art Surveys in Computational Mechanics, A. K. Noor, ed., ASME, New York, 1989, pp. 71-143.

[32] Y. MADAY, A. T. PATERA, AND E. M. RØNQUIST, *Optimal Legendre spectral element methods for the multi-dimensional Stokes problem*, in preparation.

[33] Y. MADAY AND A. QUARTERONI, *Spectral and pseudospectral approximation of the Navier-Stokes equations*, SIAM J. Numer. Anal., 19 (1982), pp. 761-780.

[34] J. F. MAÎTRE, F. MUSY, AND P. NIGON, *A fast solver for the Stokes equations using multigrid with a Uzawa smoother*, in Advances in Multi-Grid Methods, D. Braess, U. Hackbusch, and W. Trottenberg, eds., Vieweg, Braunschweig, Wiesbaden, 1985.

[35] S. A. ORSZAG, M. ISRAELI, AND M. O. DEVILLE, *Boundary conditions for incompressible flows*, J. Sci. Comput., 1 (1986), p. 75.

[36] A. T. PATERA, *A spectral element method for fluid dynamics; laminar flow in a channel expansion*, J. Comput. Phys., 54 (1984), pp. 468-488.

[37] E. M. RØNQUIST, *Optimal spectral element methods for the unsteady three-dimensional incompressible Navier-Stokes equations*, Ph.D. thesis, Dept. of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[38] E. M. RØNQUIST AND A. T. PATERA, *A Legendre spectral element method for the incompressible Navier-Stokes equations*, in Proceedings of the Seventh GAMM Conference on Numerical Methods in Fluid Mechanics, Vieweg, Braunschweig, Wiesbaden, 1988.

[39] G. SACCHI-LANDRIANI AND H. VANDEVEN, *A multidomain spectral collocation method for the Stokes problem*, Numer. Math., 58 (1990), pp. 441-464.

[40] C. STREETT, M. Y. HUSSAINI, AND Y. MADAY, *Two spectral collocation algorithms for the incompressible Navier-Stokes equations with two nonperiodic directions*, in Proc. First Internat. Conf. Indust. Appl. Math., Paris, France, 1987.

[41] R. TEMAM, *Navier-Stokes Equations. Theory and Numerical Analysis*, North-Holland, Amsterdam, 1977.

[42] H. VANDEVEN, *Compatibilité des espaces discrets pour l'approximation spectrale du problème de Stokes périodique non périodique*, M^2AN, 4 (1989), pp. 649-688.

[43] ———, *Analysis of the eigenvalues of spectral differentiation operators*, manuscript.

[44] R. VERFURTH, *A preconditioned conjugate residual algorithm for the Stokes problem*, in Multi-Grid Methods, D. Braess, W. Hackbusch, and U. Trottenberg, eds., Vieweg, Braunschweig, Wiesbaden, 1985.

[45] Y. YAMAGUCHI, C. J. CHANG, AND R. A. BROWN, *Multiple buoyancy-driven flows in a vertical cylinder heated from below*, Philos. Trans. Roy. Soc. London, A 312 (1984), pp. 519-552.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 338–348, March 1993

EXPLICIT RUNGE–KUTTA PAIRS WITH ONE MORE DERIVATIVE EVALUATION THAN THE MINIMUM*

P. W. SHARP† AND E. SMART‡

Abstract. Existing classes of (4,5), (5,6), and (6,7) pairs use the minimum number of derivative evaluations per step. This ensures that the cost of each step is minimized. However, the cost per unit step is not necessarily minimized because it may be possible to take larger steps if an extra evaluation is used. The change in efficiency when one extra evaluation per step is used is investigated for classes of (4,5), (5,6), and (6,7) pairs with the minimum number of evaluations. The aim is to show that using an extra evaluation does not necessarily increase the cost per unit step, and that in some cases the cost may decrease.

The paper begins with the derivation of the new classes. Then the new and existing classes are compared using standard functions of the free parameters. Following this, numerical comparisons of representative pairs from the new and existing classes are presented. The new (4,5) and (5,6) pairs are found to be more efficient and just as reliable as the existing (4,5) and (5,6) pairs. For the (6,7) pairs, there is little difference in efficiency and reliability.

Key words. ordinary differential equations, explicit Runge–Kutta pairs, extra stage

AMS(MOS) subject classification. 65L05

1. Introduction.

Many classes of explicit Runge–Kutta pairs have been derived for the nonstiff initial value problem

$$y' = f(x,y), \quad x \in [x_0, x_f], \quad y(x_0) = y_0, \tag{1.1}$$

where $f : \mathbf{R} \times \mathbf{R}^n \mapsto \mathbf{R}^n$.

Existing pairs up to order (6,7) use the minimum number of derivative evaluations per step. This choice ensures that the cost of each step is minimized. However, the cost per unit step is not necessarily minimized, because it may be possible to take larger steps if an extra evaluation is used on each step. This situation arises with the (7,8) pairs of Verner [8]. While 12-stage pairs exist, Verner found 13-stage pairs more efficient.

Verner's result raises interesting questions about (4,5), (5,6), and (6,7) pairs. How is the efficiency of the pairs affected when an extra evaluation is used? Can the pairs be made more efficient, as for the (7,8) pairs?

For classes of efficient (4,5), (5,6), and (6,7) pairs, we investigate the effect of adding one derivative evaluation. The pairs generate order p and order $p-1$ approximations y_{i+1}, $\widehat{y}_{i+1}$, to $y(x_{i+1})$, $i = 0, \ldots$, according to

$$y_{i+1} = y_i + h\sum_{j=1}^{s} b_j f_j, \qquad \widehat{y}_{i+1} = y_i + h\sum_{j=1}^{s} \hat{b}_j f_j,$$

where

$$f_j = f\left(x_i + c_j h, y_i + h\sum_{k=1}^{j-1} a_{jk} f_k\right), \qquad j = 1, \ldots, s.$$

*Received by the editors August 14, 1989; accepted for publication (in revised form) February 3, 1992. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada, and the Information Technology Research Centre of Ontario, Canada.

†Department of Mathematics and Statistics, Queen's University, Kingston, Canada, K7L 3N6.
‡Department of Computer Science, University of Toronto, Toronto, Canada, M5S 1A4.

We assume that the solution is advanced using the higher-order (the order p) approximation because most pairs are implemented in this mode. We also assume, as is done for most pairs, that

$$c_i = \sum_{j=1}^{i-1} a_{ij}, \qquad i = 2, \ldots, s. \tag{1.2}$$

The performance of a pair of nonstiff problems depends primarily on the truncation coefficients of the two formulae in the pair. This observation suggests we could analyze the truncation coefficients to measure the changes in performance when a stage is added. The analysis is readily performed for very low-order pairs. But the analysis is difficult for (4,5), (5,6), and (6,7) pairs because the truncation coefficients depend on the free parameters of the class in a complicated way. Fortunately, even though very low-order pairs are used infrequently, analysis of their truncation coefficients gives insight into the possible changes for higher-order pairs. Hence, to motivate our work, we perform a simple analysis for three- and four-stage (2,3) pairs.

The order conditions for an s-stage third-order formula are

$$\sum_{i=1}^{s} b_i = 1, \quad \sum_{i=2}^{s} b_i c_i = \frac{1}{2}, \quad \sum_{i=2}^{s} b_i c_i^2 = \frac{1}{3}, \quad \sum_{i=3}^{s} b_i \sum_{j=2}^{i-1} a_{ij} c_j = \frac{1}{6}. \tag{1.3}$$

A third-order formula is found by solving (1.2) and (1.3). The order conditions for the embedded second-order formula are the first two equations of (1.3) with b_i replaced by $\hat{b}_i$. These two equations are satisfied by taking $\hat{b}_i$, $i = 3, \ldots, s$, as free parameters and solving for $\hat{b}_1$ and $\hat{b}_2$. To simplify the analysis, we take $c_s = 1$.

For $s = 3$, we take c_2 as a free parameter. The principal truncation coefficients of the third-order formula are $t_1 = (1 - 2c_2)/72$, $t_2 = 1/24$, $t_3 = -3t_1$, and $t_4 = -t_2$. The minimum possible value for the maximum of the magnitude of the truncation coefficients is 1/24. The minimum is attained for $0 \le c_2 \le 1$.

For $s = 4$, we take c_2, c_3, a_{42}, a_{43}, and b_4 as free parameters. As might be expected, since four-stage fourth-order formulae exist, the principal truncation coefficients for $s = 4$ can be made arbitrarily small. This does not mean the four-stage pairs can be *arbitrarily* more efficient than the three-stage pairs, because the fifth- and higher-order terms in the error expansion must be considered. However, the four-stage pairs can be *significantly* more efficient. To illustrate this, we solve the orbit problem D5 in DETEST (Enright and Pryce [2]), using a three-stage and a four-stage pair. The coefficients of the pairs are given in the Appendix. The pairs are implemented as described in §4. We use local error tolerances of 10^{-i}, $i = 2, 3, 4, 5$. The results are summarized in Fig. 1. The four-stage pair uses fewer functions than the three-stage pair to achieve the same end-point global error.

In §2 we give the derivation of the pairs. Then in §3 we investigate the new classes using standard functions of the free parameters. We also present a representative pair from each class. In §4 we summarize numerical comparisons of the representative pairs from the new and existing classes. Finally, in §5 we briefly summarize our work and possible extensions to it.

2. Derivation.

2.1. (4,5) pairs. Most classes of (4,5) pairs use six stages. One exception is the seven-stage class of Dormand and Prince [1]. However, the seventh stage in this

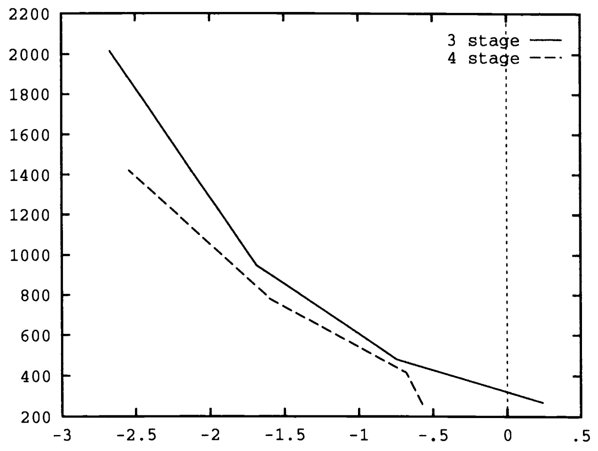


FIG. 1. *Number of derivative evaluations against the logarithm of the end-point global error for the three- and four-stage* (2,3) *pairs.*

class is reused as the first stage on the next step. Hence, the number of derivative evaluations performed on all accepted steps, except the first, is the same as for six-stage pairs. Pairs that reuse the last stage are often called FSAL (first same as last) pairs.

The numerical testing of Dormand and Prince [1] suggests that representative pairs from their FSAL class are more efficient than representative pairs from other (4,5) classes. Hence our (4,5) pairs are derived from the FSAL class. Our pairs are seven-stage non-FSAL.

We make the simplifying assumptions

$$C(i,k) \equiv \frac{c_i^{k+1}}{k+1} = \sum_{j=2}^{i-1} a_{ij} c_j^k, \quad i = 3,\dots,7, \quad k = 1,2, \tag{2.1}$$

and

$$R(j) \equiv \sum_{i=j+1}^{7} b_i a_{ij} = b_j(1-c_j), \qquad j = 1,\dots,7. \tag{2.2}$$

The simplifying assumptions imply $c_7 = 1$, $b_2 = 0$, and $\hat{b}_2 = 0$.

The independent order conditions are then

$$\sum_{i=1}^{7} b_i c_i^k = \frac{1}{k+1}, \qquad k = 0,\dots,4, \tag{2.3}$$

$$\sum_{i=1}^{7} \hat{b}_i c_i^k = \frac{1}{k+1}, \qquad k = 0,\dots,3, \tag{2.4}$$

$$\sum_{i=3}^{7} b_i c_i a_{i2} = 0, \tag{2.5}$$

$$\sum_{i=3}^{7} \hat{b}_i a_{i2} = 0. \tag{2.6}$$

These equations and the simplifying assumptions are solved as follows:

- Take b_7 as a free parameter and solve (2.3) for b_i, $i = 1, 3, \ldots, 6$.
- Solve $C(3,1)$, $C(3,2)$ for a_{32} and c_2, and $C(4,1)$, $C(4,2)$ for a_{42} and a_{43}.
- Take a_{52} as a free parameter and solve $C(5,1)$, $C(5,2)$ for a_{53} and a_{54}.
- Using $c_7 = 1$ and $R(2)$, (2.5) can be rewritten as

$$\sum_{i=3}^{6} b_i(1 - c_i)a_{i2} = 0.$$

Solve this for a_{62} and then (2.5) for a_{72}.

- Take a_{65} as a free parameter and solve $C(6,1)$ and $C(6,2)$ for a_{63} and a_{64}.
- Solve $R(6)$ for a_{76} and $R(5)$ for a_{75}.
- Solve $C(7,1)$ and $C(7,2)$ for a_{73} and a_{74}.
- Solve (1.2) for $a_{i1}, i = 2, \ldots, 7$.
- Take $\hat{b}_7$ as a free parameter and solve (2.4) and (2.6) for $\hat{b}_i$, $i = 1, 3, 4, 5, 6$.

The remaining simplifying assumptions are automatically satisfied. The derivation gives a class of seven-stage (4,5) pairs with c_3, c_4, c_5, c_6, a_{52}, a_{65}, $\hat{b}_7$, and b_7 as free parameters.

2.2. (5,6) pairs. Most classes of (5,6) pairs use eight stages. However, recently, several investigators (see Verner [9], for example) derived similar classes of nine-stage (5,6) FSAL pairs. The numerical testing of these investigators suggests that representative pairs from the new classes are more efficient than representative pairs from eight-stage classes. Hence we derive our (5,6) pairs from the FSAL class. Our pairs are ten-stage FSAL.

We make the simplifying assumptions $C(i,1)$, $i = 3, \ldots, 9$, $C(i,2), i = 4, \ldots, 9$, $R(j)$, $j = 1, \ldots, 9$, $a_{i2} = 0$, $i = 4, \ldots, 9$, $\hat{b}_3 = 0$, $b_3 = 0$, and $b_9 = 0$.

The simplifying assumptions imply $c_9 = 1$, $b_2 = 0$, and $\hat{b}_2 = 0$.

The independent order conditions are then

$$\sum_{i=1}^{9} b_i c_i^k = \frac{1}{k+1}, \qquad k = 0, \ldots, 5, \tag{2.7}$$

$$\sum_{i=1}^{10} \hat{b}_i c_i^k = \frac{1}{k+1}, \qquad k = 0, \ldots, 4, \tag{2.8}$$

$$\sum_{i=4}^{9} b_i c_i a_{i3} = 0, \tag{2.9}$$

(2.10) $$\sum_{i=4}^{10} \hat{b}_i a_{i3} = 0,$$

(2.11) $$\sum_{i=4}^{9} b_i c_i \sum_{j=2}^{i-1} a_{ij} c_j^3 = \frac{1}{24},$$

(2.12) $$\sum_{i=4}^{10} \hat{b}_i \sum_{j=2}^{i-1} a_{ij} c_j^3 = \frac{1}{20}.$$

These equations and the simplifying assumptions are solved as follows:

• Take b_4 as a free parameter and solve (2.7) for b_1, b_i, $i = 5, \ldots, 9$.

• Solve $C(3,1)$ for a_{32}, $C(4,1)$, $C(4,2)$ for a_{43} and c_3, and $C(5,1)$, $C(5,2)$ for a_{53} and a_{54}.

• Take a_{65} as a free parameter and solve $C(6,1)$, $C(6,2)$ for a_{63} and a_{64}.

• Take a_{75} and a_{76} as free parameters and solve $C(7,1)$, $C(7,2)$ for a_{73} and a_{74}.

• Using $c_9 = 1$ and $R(2)$, (2.9) can be rewritten as

$$\sum_{i=4}^{8} b_i(1 - c_i)a_{i3} = 0.$$

Solve this for a_{83}.

• Rewrite (2.11) as

$$\sum_{i=4}^{8} b_i(c_i - 1)q_{i3} = -\frac{1}{120},$$

where $q_{i3} = \sum_{j=2}^{i-1} a_{ij}c_j^3$. Solve this for q_{83}.

• Take a_{87} as a free parameter and solve $C(8,1)$, $C(8,2)$, $q_{83} = \sum_{j=2}^{7} a_{8j}c_j^3$ for a_{84}, a_{85}, and a_{86}.

• Solve $R(j)$, $j = 3, \ldots, 8$ for a_{9j}, $j = 3, \ldots, 8$.

• Set $c_{10} = 1$, $a_{10j} = b_j$, $j = 1, \ldots, 9$.

• Solve (1.2) for $a_{i1}, i = 2, \ldots, 9$.

• Take $\hat{b}_8$ as a free parameter, and solve (2.8), (2.10), and (2.12) for $\hat{b}_1$, $\hat{b}_i, i = 4, 5, 6, 7$, $\hat{b}_9$, and $\hat{b}_{10}$.

2.3. (6,7) pairs. Few numerical comparisons of (6,7) pairs have been made. Hence it is difficult to decide, using the efficiency of representative pairs, which class of pairs to base our pairs on. Instead, we use the most general class of pairs, which is that derived by Verner [8].

Verner derives his pairs by satisfying a set of conditions which ensure that a pair has the required order. The central idea of the derivation is to specify a vector of stage orders for the seventh-order formula. Once this is done, the seventh-order formula is found by solving recurrences and systems of linear equations. The sixth-order formula is then found by adding a stage and solving a recurrence and a system of linear equations. The details of the derivation are given in Verner [8]. Here, we discuss the selection of the vector of stage orders.

The seventh-order formula in Verner's (6,7) pairs uses the first nine stages. The vector of stage orders for the formula is

$$\xi = [7, 1, 2, 3, 3, 3, 3, 3, 3].$$

The seventh-order formula of our pairs uses the first ten stages, which means that one component must be added to ξ. There are two restrictions on the component: it must be less than three, and it must be added between the first and fourth component of ξ. Even with these restrictions the value and position of the component is not unique. We experimented with two choices and decided upon

$$\xi = [7, 1, 1, 2, 3, 3, 3, 3, 3, 3].$$

3. Selection. In this section, we summarize the selection of representative pairs from the new classes. The selection uses standard functions of the free parameters. Only a brief description of the functions is given here. See Prince and Dormand [5] or Shampine [6] for a more detailed description.

(i) The truncation coefficients in the higher-order formula should be small or close to their minimum value to make the pair efficient. We define the size of the truncation coefficients as

$$T_{p+1} = \left[\sum_{j=1}^{N_{p+1}} t_{p+1,j}^2 \right]^{1/2},$$

where N_{p+1} is the number of order conditions of order $p+1$, and $t_{p+1,j}$ is the jth truncation coefficient of order $p+1$. Small truncation coefficients will also lead to small errors in the local error estimate.

(ii) The ratio

$$R^{\text{opt}} \equiv \frac{\left[\sum_1^{N_{p+1}} (t_{p+1,j} - \hat{t}_{p+1,j})^2 \right]^{1/2}}{\hat{T}_p}$$

should be small. This helps to ensure that the local error estimate behaves like Ch^p. This in turn helps to ensure that the number of rejected steps is small.

(iii) The coefficients of the pair should not be large in magnitude to ensure that the roundoff error is insignificant (except possibly near limiting precision). We denote the maximum of the magnitude of the coefficients by $R^{\max}$. It is difficult to decide upon a suitable value for $R^{\max}$, because a small bound can prevent the selection of an efficient and reliable pair. After some numerical experiments, we chose $R^{\max} = 20$.

(iv) The region of absolute stability for the order p formula should be large. We measure the size, denoted by R^{stab}, along the negative real axis.

3.1. (4,5) pairs. As we might expect from the existence of seven-stage sixth-order formulae, the principal truncation coefficients of the fifth-order formula can be zero. This means we cannot search for an efficient pair by minimizing T_6. We minimize T_6 subject to $T_7 \leq kT_6$, $k > 0$. There is no obvious value for k. We found $k = 4$ led to useful pairs. For values of the free parameters that satisfy the above requirement, the ratio R^{opt} is usually between 1 and 1.5, and $R^{\max}$ is less than ten.

A representative seven-stage pair has $c_3 = 8/35$, $c_4 = 9/20$, $c_5 = 2/3$, $c_6 = 7/9$, $a_{52} = -7/20$, $a_{65} = 9/25$, $b_7 = 3/40$, and $\hat{b}_7 = 9/125$, for which $T_6 = 7.1 \times 10^{-5}$, $T_7 = 1.8 \times 10^{-4}$, $R^{\text{opt}} = 1.1$, $R^{\text{stab}} = -3.9$, and $R^{\max} = 0.86$.

3.2. (5,6) pairs. In a similar way to the (4,5) pairs, T_8 can be made arbitrarily small. A representative pair has $c_2 = 1/18$, $c_4 = 1/6$, $c_5 = 89/200$, $c_6 = 19/39$, $c_7 = 74/95$, $c_8 = 8/9$, $a_{65} = 8/61$, $a_{75} = -43/21$, $a_{76} = 83/33$, $a_{87} = 13/23$, $b_4 = 16/61$, and $\hat{b}_8 = 0$, for which $T_7 = 3.2 \times 10^{-6}$, $T_8 = 2.4 \times 10^{-5}$, $R^{\text{opt}} = 1.0$, $R^{\text{stab}} = -4.5$, and $R^{\max} = 13.3$.

3.3. (6,7) pairs. Adding one stage does not lead to a large reduction in the size of the truncation coefficients for the seventh-order formula. We find that T_8 can be made approximately three times smaller. A representative 11-stage pair is given by $c_2 = 1/50$, $c_3 = 27/125$, $c_4 = 41/100$, $c_5 = 57/100$, $c_6 = 43/100$, $c_8 = 18/25$, $c_9 = 5/6$, $a_{42} = 0$, and $a_{61} = -31/200$, for which $T_8 = 1.3 \times 10^{-5}$, $T_9 = 3.6 \times 10^{-5}$, $R^{\text{opt}} = 1.9$, $R^{\text{stab}} = -3.8$, and $R^{\max} = 9.4$.

4. Numerical comparisons. In this section, we summarize numerical comparisons of existing (4,5), (5,6), and (6,7) pairs with the pairs from the previous section. To help ensure fair comparisons, we have sought to use existing pairs that are efficient. This requirement presents little difficulty for (4,5) and (5,6) pairs. We use the (4,5) FSAL pair of Dormand and Prince [1] and the (5,6) FSAL pair of Verner [9]. The difficulty comes with the (6,7) pairs, because existing (6,7) pairs are either inefficient or unreliable. The (6,7) pair given in Verner [8] is for illustrative purposes only and is inefficient. The (6,7) pair of Fehlberg [3] has an unreliable local error estimate. To overcome these difficulties, we selected, using the criteria of §3, a pair from the ten-stage (6,7) class of Verner [8]. The values for the free parameters are $c_2 = 1/5$, $c_4 = 17/40$, $c_5 = 2/25$, $c_7 = 7/9$, and $c_8 = 5/6$. The remaining coefficients are found using Theorems 3 and 4 of Verner [8].

We use the test problems A1,...,A5, B1,...,B5, D1,...,D5, E1,...,E5 from DETEST (Enright and Pryce [2]). The problems are integrated in their scaled form and we refer to them as problems 1–20. We omit the C set of problems because the set contains mildly stiff problems.

The pairs are implemented in a similar way to the pair in DVERK (Hull, Enright, and Jackson [4]). After an accepted step, x and y are updated and a new stepsize is selected according to the formula

$$\min\left\{\alpha h_{\text{old}}, \beta(\text{TOL}/\text{est})^{1/p} h_{\text{old}}, h_{\max}\right\},$$

where $\alpha > 1$, β is a safety factor, TOL is the local error tolerance, est is the weighted norm of the local error estimate, $h_{\max}$ is the maximum stepsize allowed, and h_{old} is the previous stepsize. If the step is rejected, a new stepsize is calculated in one of two ways, depending on the number of consecutive rejected steps (denoted by n_r). If $n_r \leq m_{\text{opt}}$ where $m_{\text{opt}} \geq 0$, the new stepsize is calculated as

$$\max\left\{\alpha^{-1} h_{\text{old}}, \beta(\text{TOL}/\text{est})^{1/p} h_{\text{old}}, h_{\min}\right\},$$

where $h_{\min}$ is the minimum stepsize. For $n_r > m_{\text{opt}}$, the new stepsize is the maximum of $h_{\min}$ and $\alpha^{-1} h_{\text{old}}$. We use a maximum norm with absolute weights, $\alpha = 2$, $\beta = 0.9$, $h_{\max} = 5$, $h_{\min} = 10^{-8}$, and $m_{\text{opt}} = 1$. For each pair and test problem we use tolerances of 10^{-i}, $i = 2, \ldots, 10$.

We begin with comparisons of the efficiency. The comparisons are based on the normalized efficiency statistics printed by DETEST. The package calculates the statistics, for a given problem and pair, by first assuming a relationship of the form

$$\text{global error} = \text{CTOL}^{E},$$

where the global error is either the end point or maximum global error. The exponent E and the constant of proportionality C depend on the method and the problem. After the global error is found for several tolerances, values for E and C are found in a least squares sense. The values are then used to estimate the number of derivative evaluations required to achieve an expected accuracy (global error). The number of derivative evaluations is estimated for a range of expected accuracies to give the normalized efficiency statistics.

Once the statistics have been obtained, we take the pairs two at a time and form the intersection of the values of the expected accuracy for each problem. For these values, we take the estimated number of derivative evaluations and divide the larger value by the smaller value to give an efficiency ratio. We then form the efficiency gain by subtracting 1 from the ratio. If the second pair of the two pairs uses fewer evaluations, we multiply the gain by -1. Finally, we multiply the gains by 100, round the result to the nearest integer, and enter the numbers on a graph of the expected accuracy against the problem number. Efficiency gains of magnitude greater than 98 are entered as 99 (with the appropriate sign). See Sharp [7] for a more detailed discussion of efficiency gains.

Figures 2, 3, and 4 contain the efficiency gains for the new (4,5), (5,6), and (6,7) pairs against the existing pairs of the same order. Figures 5 and 6 contain the efficiency gains for the new (4,5) and (5,6) pairs against the new (6,7) pair. Table 1 contains the average of the efficiency gains for each problem in Figs. 2–6. The end point global error is used in the normalized statistics.

The new (4,5) pair is often significantly more efficient than the Dormand and Price (4,5) FSAL pair. The average gain in efficiency over all 20 problems and 9 tolerances is approximately 35 percent. The new (5,6) pair is often more efficient than Verner's (5,6) FSAL pair. But, the gain in efficiency is not as significant as for the (4,5) pair. The two (6,7) pairs are of comparable efficiency. The new (6,7) pair is significantly more efficient than the new (4,5) pair, but of comparable efficiency to the new (5,6) pair.

											-60				-17					
-2			-14			-16			-69	-53	-39	-50	-40	-38	-12		-32			
			-12		12	-31			-57	-40	-37	-40	-32	-32	-10	-50	-39	-99		
-4			2		0	-42	-4	-21	-40	-52	-44	-34	-29	-28	-11	-51	-46	-99	12	
	-50	-34	9	-5	-2	-43	-12	-25	-38	-44	-46	-28	-22	-20	-1	-48	-54	-99	-5	22
-6	-53	-56	18	-62	-39	-42	-14	-41	-36	-23	-42	-44	-17	-7	1	-46	-52	-99	2	37
	-53	-93	27	-83	-60	-48	-21	-46	-26	-13	-41	-56	-25	-12	-3	-44	-47	-89	-9	3
-8	-46	-99	38	-69	-80	-63	-28	-60	-14	-10	-42	-59	-26	-10		-40	-48	-84	-23	2
	-38	-78	55	-57	-94		-38	-75	-6	-9						-35	-53	-78	-24	11
-10	-30	-67		-44			-42	-89	0										-28	-13
	-21						-44													

FIG. 2. *Efficiency gains for the Dormand and Prince* (4,5) *pair against the new* (4,5) *pair.*

One measure of the reliability of a pair is the accuracy of the local error estimate. Table 2 contains, for each problem and pair, the maximum of the norm of the local error on accepted steps divided by the tolerance (TOL). For example, the first entry for the Dormand and Price (4,5) pair is 0.6. This means, for problem one, that the norm of the local error on accepted steps for all nine tolerances was never greater than 0.6TOL.

			12	
-2		5 -1	19 -4 18 18	
	-34	18 6-26	8-10 11 9	-19-2 -6
-4	-59 22	14 16 0-31	6-18 21 6	-6 0-33
	4 12-51 12 5	-1 15 5-20-25	0-15 12 3	0 8-36 12 12
-6	6 12-56-13 9	-22 14-10-37-25	-9-10-10 1	5 8-53 4 7
	6 16-51 17-10	-34 15 -2-55-24	-20-16-11-3	12 3-68 8-26
-8	-6 9-47 13-10	-45 10 -7 -23	-33-16-65	21 1-83 17-59
	-14 -3-47 17-23	-3 -4		33 5 17 -5
-10	-20-18 12	-8 -2		15
	-20	-16 -1		
-12		-1		

FIG. 3. *Efficiency gains for Verner* (5,6) *pair against the new* (5,6) *pair.*

			13 23	
-2		11	-2 14 22 23 1	30
		17 -10 -2	-4 7 28 14 12	-6 25 2
-4	19 10	1 -25 12	-8 0 23 13 12	0 3 7
	0 12 10 10	-2 7-19-29 7	-23 1 20 10 6	0 2 11 10 10
-6	-20 14 2 10 4	-7 2-13-30 2	-36 2 18 4 2	0-10 2 -1 10
	-17 31 0 -7 1	0 0-12-29 10	-40 1 16 -1 0	2-13 4-21-43
-8	-21 14-11 -8 -7	5 -1-10-29 18	-43-11 9 -5 -2	4-21 0-12-37
	-13 4-17 5 -2	4 -2-10-30 17	-44-20 0	5-30 3 -5-31
-10	-7 5 6	-6 -7 23		7-38 8 5-20
	0 12 1	-9 -1		12
-12	4	-10 1		

FIG. 4. *Efficiency gains for Verner* (6,7) *pair against the new* (6,7) *pair.*

0			77	
			0 16 21 42 43	
-2	2	-10	-6 11 1 15 7	29-10
	-17	2 -24-10	-20 2 -2 -2 -2	-17 0-64
-4	49-22 52	-16 -34-24	-40 -8-15-14-15	-21-17-52
	6-27 48 25	-35 33-19-52-32	-66-13-22-31-24	-38-38-67 44 77
-6	-20 -8-25 -3-13	-43 23-37-69-47	-90-32-29-30-22	-58-54-88 12 37
	-29-32-34-76-43	-50 7-61-81-62	-99-62-53-50-39	-77-66-96-29-52
-8	-38-73-54-99-60	-69 -4-84-90-87	-99-99-87-81	-98-89-99-40-66
	-40-99-69-97-69	-26-99-99-99		-99-99-99-65-14
-10	-51-99 -99	-49-99-99		-86-34
	-55	-72		

FIG. 5. *Efficiency gains for the new* (4,5) *pair against the new* (6,7) *pair.*

0			94	
			51 22 16 30 26	
-2	-13	6	46 22 0 20 10	60 84
	-17	26 -9 2	25 9 0 -3 2	-2 32 4
-4	58-24 37	16 -33 -3	14 0 -2 -4 -4	6 11 -1
	19-29 37 7	-1 33 1-56 -9	-1 -3-10-11-14	6 0-18 37 37
-6	10 23-28 7 4	-10 32 3-90-19	-20 0-14-21-24	6-10-40 27 5
	14 19-28 -3-15	-8 28 4-99-27	-33 -7-13-20-22	6-21-56 -3-58
-8	3 -2-45-45-25	-15 21 6 -32	-47-28-23-17	8-31-90 -2-70
	7-29-56-33-32	9 7		6-47 -9 -3
-10	4-41 -45	-1 8		5 -8
	3	-8 10		
-12		11		

FIG. 6. *Efficiency gains for the new* (5,6) *pair against the new* (6,7) *pair.*

TABLE 1
The average of the efficiency gains in Figs. 2–6.

(4,5)/(4,5)	-41 -71 15 -53 -37	-40 -25 -51 -31 -30	-43 -44 -27 -21 -7	-44 -46 -92 -10 10
(5,6)/(5,6)	-6 4 -49 9 -1	-9 5 -2 -17 -25	-4 -12 0 2 6	6 3 -46 12 -14
(6,7)/(6,7)	-10 11 0 2 2	2 -2 -8 -21 10	-20 0 17 8 6	1 -10 7 -1 -18
(4,5)/(6,7)	-38 -36 -30 -54 -18	-35 -12 -66 -62 -51	-52 -23 -23 -18 3	-58 -41 -71 -27 -8
(5,6)/(6,7)	6 6 -30 -13 -4	1 16 6 -46 -14	4 1 -5 -3 8	5 0 -16 7 -17

TABLE 2
The maximum of the norm of the local error on accepted steps, divided by TOL.

DP(4,5)	0.6 7.9 3.5 2.0 0.7	5.7 0.6 1.4 12.4 6.2	9.5 11 8.6 6.8 2.3	1.3 26 13 0.9 0.9
new (4,5)	0.6 0.8 1.1 1.0 1.0	9.3 0.8 0.9 1.8 1.3	0.9 1.0 1.5 1.6 2.4	1.1 1.3 1.2 1.0 0.9
V(5,6)	1.1 2.0 3.4 1.4 0.8	6.7 3.8 0.8 3.7 2.2	2.5 5.3 5.0 9.8 8.5	1.4 4.1 3.0 1.3 0.9
new (5,6)	1.2 0.4 1.0 1.0 0.5	1.0 0.8 0.9 1.0 1.0	1.0 1.0 1.0 1.0 1.1	0.7 1.0 1.0 1.0 1.0
V(6,7)	0.5 0.9 1.1 0.9 0.6	1.1 0.7 0.8 1.1 1.0	0.9 1.0 1.0 1.0 1.2	0.8 1.2 1.0 0.9 1.0
new (6,7)	0.5 0.7 1.1 0.9 0.8	1.6 0.5 0.8 1.0 1.0	1.1 2.8 2.3 1.0 1.6	0.8 1.2 1.0 0.9 1.0

If the local error estimate is accurate, the maximum norm should be no larger than TOL. (Because of the conservative nature of the stepsize selection, the maximum norm may be smaller than TOL.)

The new (4,5) and (5,6) pairs have a significantly more accurate local error estimate than the existing (4,5) and (5,6) pairs. However, there is little difference between the reliability of the two (6,7) pairs.

5. Discussion.

We derived classes of (4,5), (5,6), and (6,7) explicit Runge–Kutta pairs that use one more derivative evaluation per step than existing classes. We also tested representative pairs from the new and existing classes. We found the new (4,5) and (5,6) pairs were more efficient and just as reliable as the existing (4,5) and (5,6) pairs. The new and existing (6,7) pairs were of comparable efficiency and reliability.

Besides increasing the number of classes of suitable pairs for implementation in general-purpose software, our results have several possible implications. Because of the similarities between Runge–Kutta pairs and Nyström pairs for the general second-order initial-value problem, our conclusions may hold for Nyström pairs. For instance, an eight-stage (5,6) Nyström pair is easily obtained by applying a simple transformation to the coefficients of an eight-stage (5,6) Runge–Kutta pair. Hence it may be possible to improve the efficiency of the (5,6) Nyström pairs by adding a stage.

If interpolants for the new classes require no more extra stages than those for existing classes, then the relative cost of forming the interpolant will be less. In this case, the cost of generating dense output and using defect control will be less. We are currently investigating interpolants for the new classes.

Appendix.

The appendix contains the coefficients of the three- and four-stage (2,3) pairs used to produce the numerical results of §1.

TABLE A.1
Tableau for the three-stage (2, 3) *pair.*

$\frac{1}{3}$	$\frac{1}{3}$		
1	-1	2	
$\hat{b}$	$\frac{1}{2}$	0	$\frac{1}{2}$
b	0	$\frac{3}{4}$	$\frac{1}{4}$

TABLE A.2
Tableau for the four-stage (2, 3) *pair.*

$\frac{1}{4}$	$\frac{1}{4}$			
$\frac{3}{5}$	$-\frac{387}{625}$	$\frac{762}{625}$		
1	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{3}{5}$	
$\hat{b}$	$\frac{1}{10}$	$\frac{2}{5}$	$\frac{1}{4}$	$\frac{1}{4}$
b	$-\frac{1}{90}$	$\frac{8}{15}$	$\frac{5}{18}$	$\frac{1}{5}$

Acknowledgment. The authors thank the referees, who suggested three ways of improving our paper.

REFERENCES

[1] J.R. Dormand and P.J. Prince, *A family of embedded Runge–Kutta formulae*, J. Comput. Appl. Math., 6 (1980), pp. 19–26.

[2] W.H. Enright and J.D. Pryce, *Two* FORTRAN *packages for assessing initial value methods*, Trans. Math. Software, 13 (1987), pp. 1–27.

[3] E. Fehlberg, *Classical fifth, sixth, seventh, and eighth order Runge–Kutta formulas with stepsize control*, Technical Rep. TR R-287, NASA, 1968.

[4] T.E. Hull, W.H. Enright, and K. R. Jackson, *User's guide for* DVERK*—a subroutine for solving nonstiff* ODE*'s*, TR 100/76, Dept. of Computer Science, Univ. of Toronto, Canada, 1976.

[5] P.J. Prince and J.R. Dormand, *High order embedded Runge–Kutta formulae*, J. Comput. Appl. Math., 7 (1981), pp. 67–76.

[6] L.F. Shampine, *Some practical Runge–Kutta formulas*, Math. Comp., 46 (1986), pp. 135–150.

[7] P.W. Sharp, *Numerical comparisons of some explicit Runge–Kutta pairs of orders* 4 *through* 8, Trans. Math. Software, 17 (1991), pp. 387–409.

[8] J.H. Verner, *Explicit Runge–Kutta methods with estimates of the local truncation error*, SIAM J. Numer. Anal., 15 (1978), pp. 772–790.

[9] ———, *Some Runge–Kutta formula pairs*, SIAM J. Numer. Anal., 28 (1991), pp. 496–511.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 349–367, March 1993

SUPERPARALLEL FFTS*

HANS MUNTHE-KAAS†

Abstract. Fast Fourier transform (FFT) algorithms for single instruction multiple data (SIMD) machines are developed which simultaneously solve any combination of FFTs of different sizes and even different spatial dimensionalities. The only restrictions are that all the periods must be powers of two and that the initial data must satisfy some alignment requirements with the address space in the computer. The degree of parallelism is equal to the sum of the sizes of all the subproblems and the (parallel) solution time is proportional to $\log_2(m)$, where m is the number of points in the largest subsystem. It is shown that the task of unscrambling the data can be both executed and scheduled efficiently in parallel. Finally, implementations on the MasPar computer are described. The codes can be quickly and easily employed in solving complicated problems, and the interface for the routines may therefore be interesting for sequential FFT codes as well.

Key words. FFT, parallel computing, SIMD computers, superparallel algorithms

AMS(MOS) subject classifications. 65T20, 65Y05, 65Y10

1. Introduction.

Fast Fourier transforms (FFTs) constitute one of the most important classes of algorithms in scientific computations. Several papers deal with the problems of implementing FFTs on vector and multiprocessor systems; see [1], [26], and [27]. Various aspects of mapping FFT algorithms to Boolean cubes are discussed, for example, in [11] and [23].

In this paper we are concerned with the effective implementation of FFTs on massively parallel computer systems. Primarily, we have in mind fine-grained parallel machines of the single instruction multiple data (SIMD) stream type, often also called *data parallel computers.* SIMD machines are now commercially available with a number of processors ranging from a few thousand to almost a hundred thousand. Increasingly, we are faced with the luxury of having "too many processors."

Suppose that we wish to solve a mathematical problem of some characteristic size n (n is, e.g., the number of points in a grid, the number of pixels in an image, or some other size indicator). Parallel algorithms for solving the problem can typically be grouped into three main categories.

1. $p \ll n$. The number of processors is much smaller than the size of the problem. This is a typical situation for implementations on coarse-grained parallel systems with a moderate number of processors.
2. $p \sim n$. The number of processors matches the size of the problem. This is not as common in practice; it mainly occurs for benchmark model problems or for special-purpose machines built for a specific task.
3. $p \gg n$. The number of processors is much larger than the size of the problem. This problem class is becoming increasingly important as the number of processors grows, and is our concern in this paper.

As the maximum number of processors that can possibly be employed in solving a given problem is limited by (some function of) n, it is not likely that problem class 3 has efficient algorithms in the case where only a single instance of the problem is to be solved. On the other hand, it is often the case that we want to simultaneously

*Received by the editors May 22, 1991; accepted for publication (in revised form) February 27, 1992. This work was sponsored in part by the Norwegian Science Foundation NAVF through the project "Partielle Differensiallikninger i Storskalaberegninger" and by the University of Bergen.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. A large portion of this work was done while the author was a visiting researcher at Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS), Toulouse, France.

solve multiple occurrences of the same (or closely related) problems. Parkinson [22] uses the term *superparallel algorithms* to denote algorithms that in a SIMD fashion can solve multiple instances of similar problems (of possibly different sizes), with a degree of parallelism that is on the order of the sum of the sizes of all the subproblems. Superparallel algorithms are immensely important for the efficient utilization of massively parallel SIMD machines, as they in a sense make the SIMD machine behave as a MIMD (multiple instruction multiple data) stream machine, solving different problems simultaneously. Compared to a parallel scheduling of different jobs on a MIMD computer, superparallel algorithms have several advantages, since they require no load balancing or synchronizations.

Before we describe the superparallel FFT algorithms in detail, we will briefly address the main differences between sequential and data parallel programming, since this is a key issue for understanding the fundamental ideas. In a sequential programming style, the geometry information of a problem is typically controlled through the instruction stream (i.e., through the program code). For instance, in grid problems the size of the problem usually appears in the range of some looping variables. Data parallel programming [8], on the other hand, seeks to define geometries through the data streams. For example, grid problems are defined by mapping the grid onto the processors, and boundary effects are taken into account by modifying the data set near the boundary points. Since SIMD machines have many data streams, but only one instruction stream, defining the geometries through the data stream is the only way of obtaining superparallel algorithms.

Superparallel algorithms were previously known for problems such as the solution of linear tridiagonal equation systems and linear recurrences [22]. Superparallel FFT algorithms have, on the other hand, not been known. The reason for this is probably that it is less obvious how to define the geometries of FFTs through the data stream. Parkinson [22] writes: "[For the FFT algorithm], there does not appear to be a variable which would allow us to easily extend the algorithm to have the Super Parallel property."

In the following sections we will demonstrate that such a variable exists, and construct superparallel FFT algorithms. It came as a pleasant surprise that in many ways the corresponding codes are both simpler to program and simpler to use than their sequential counterparts.

2. The superparallel FFT.

2.1. Background and fundamental ideas.

Given an n-periodic vector

$$x(p+n) = x(p); \qquad p = 0, 1, \ldots, n-1,$$

the domain of this vector may be identified with the n-cyclic group $\mathcal{G} = \mathcal{Z}_n$, i.e., the integers $0, 1, \ldots, n-1$ under addition modulo n.

The Fourier transform on $\mathcal{Z}_n$ is defined as

$$\hat{x} = \sum_{q \in \mathcal{G}} x(q) \cdot e^{2\pi i p q/n},$$

or equivalently, in matrix form,

$$\hat{x} = F_n \cdot x,$$

where F_n is the Fourier matrix of order n

$$(F_n)_{p,q} = e^{2\pi i p q/n}. \tag{1}$$

The domain of $\hat{x}$ is the dual group $\hat{\mathcal{G}}$, which is isomorphic to $\mathcal{G}$. Or, in simpler language, $\hat{x}$ is also an n-periodic vector, and there is a natural 1–1 correspondence between the elements of $\hat{x}$ and the elements of x.

Now given a general finite Abel group $\mathcal{G}$, it can always be written as a direct product of cyclic groups $\mathcal{Z}_{n_i}$

$$\mathcal{G} = \mathcal{Z}_{n_{k-1}} \otimes \mathcal{Z}_{n_{k-2}} \otimes \cdots \otimes \mathcal{Z}_{n_1} \otimes \mathcal{Z}_{n_0}.$$

$\mathcal{G}$ can be identified with the set of all integer k-tuples

$$(l_{k-1}, l_{k-2}, \ldots, l_0); \qquad 0 \le l_i < n_i - 1,$$

under addition modulo $(n_{k-1}, n_{k-2}, \ldots, n_0)$. One can simply think of $\mathcal{G}$ as being a k-dimensional grid, periodic of period n_i in the direction i. The Fourier transform on $\mathcal{G}$ is given by

$$\hat{x}(p) = \sum_{q \in \mathcal{G}} x(q) \cdot e^{2\pi i \langle p,q \rangle / n},$$

where $\langle p, q \rangle$ is the bilinear pairing

$$\langle p, q \rangle = \frac{p_{k-1} q_{k-1}}{n_{k-1}} + \frac{p_{k-2} q_{k-1}}{n_{k-2}} + \cdots + \frac{p_1 q_1}{n_1} + \frac{p_0 q_0}{n_0}.$$

The transform can equivalently be written in matrix form

$$\hat{x} = (F_{n_{k-1}} \otimes F_{n_{k-2}} \otimes \cdots \otimes F_{n_0}) \cdot x,$$

where F_n is given by (1) and $\otimes$ denotes the matrix tensor product (see, e.g., [7] for its definition and properties). The domain of $\hat{x}$ is the dual group $\hat{\mathcal{G}}$, which again is isomorphic to the original group $\mathcal{G}$.

In the rest of the paper we make the assumption that all periods n_i are powers of two. It is our aim to develop superparallel FFTs that can simultaneously handle any collection of FFTs of (possibly) different sizes and different number of dimensions k, the only restriction being that all the periods are powers of two. Our approach is based on Cooley–Tukey FFTs [6].

The idea behind the radix-2 Cooley–Tukey FFT is an attempt to factor the group $\mathcal{Z}_n$ in a direct product of binary groups $\mathcal{Z}_2$. But since $\mathcal{Z}_n$ and $\mathcal{Z}_2 \otimes \mathcal{Z}_{n/2}$ are in general nonisomorphic groups, we cannot factor the matrix F_n as $F_2 \otimes F_{n/2}$. The closest we can get to a factorization of this kind is the following formula (described in §2.2):

$$P \cdot F_n = (I_2 \otimes F_{n/2}) \cdot T \cdot (F_2 \otimes I_{n/2}), \tag{2}$$

where P is a permutation matrix, called the *scrambling matrix.* I_k are identity matrices of order k, and T is a diagonal matrix containing the so-called *twiddle factors.*

Since $(I_2 \otimes F_{n/2}) \cdot (F_2 \otimes I_{n/2}) = (F_2 \otimes F_{n/2})$ is the FFT on the group $Z_2 \otimes Z_{n/2}$, we observe that apart from the twiddle factors T, and the scrambling matrix P, the FFTs on $\mathcal{Z}_n$ and $\mathcal{Z}_2 \otimes \mathcal{Z}_{n/2}$ are identical. We can continue to factor out binary groups and arrive at an equivalency between the FFT on

$$\mathcal{Z}_{n_{k-1}} \otimes \mathcal{Z}_{n_{k-2}} \otimes \cdots \otimes \mathcal{Z}_{n_1} \otimes \mathcal{Z}_{n_0}$$

and on the d-dimensional binary group

$$\mathcal{Z}_2 \otimes \mathcal{Z}_2 \otimes \cdots \otimes \mathcal{Z}_2,$$

where $d = \log_2(n_{k-1}\cdot n_{k-2}\cdots n_0)$, the only differences between the two transforms being the values of the twiddle factors and the scrambling of the results.

Thus, since all the geometric information about the transforms is contained in the twiddle factors (and in the scrambling of the results), it is possible to code the geometric information of FFT*s in the data stream instead of in the instruction stream. This is the basis for the development of superparallel* FFT*s.*

2.2. The basic algorithm. To give a precise meaning to the splitting formula (2), we must define a matching between the elements of different groups. This is done through the *bit representation* of the elements of a group, defined by stacking together the binary representation of each component of the group element.

For example, suppose we want to identify the elements of a group $\mathcal{G}_1 = \mathcal{Z}_8\otimes\mathcal{Z}_4\otimes\mathcal{Z}_4$ with the elements of the group $\mathcal{G}_2 = \mathcal{Z}_2\otimes\mathcal{Z}_4\otimes\mathcal{Z}_4\otimes\mathcal{Z}_4$. An element, say $g = (5,1,2) \in \mathcal{G}_1$, has the bit representation $(1,0,1,\ 0,1,\ 1,0)$. (Note that the second component is represented by the *two bits* $0,1$ since two bits are required to represent all numbers in $\mathcal{Z}_4$.) This element is identified with the element $(1,1,1,2) \in \mathcal{G}_2$, which has the same bit representation.

Thus the bit representation identifies a group element in $\mathcal{G}$ with an element of the binary group

$$\mathcal{G}_b = \mathcal{Z}_2\otimes\cdots\otimes\mathcal{Z}_2,$$

with the same cardinality as $\mathcal{G}$.

The binary representation will be the most used component-wise representation of a group element, *so from now on, unless otherwise stated,* $(g_{k-1}, g_{k-2}, \ldots, g_0)$ *refers to the bit representation of a group element* g.

The *perfect shuffle permutation* of the elements in a cyclic group $\mathcal{Z}_n$ is given by the transformation

$$\sigma(g) = \sigma((g_{k-1}, g_{k-2}, \ldots, g_0)) = (g_{k-2}, g_{k-3}, \ldots, g_0, g_{k-1}),$$

i.e., a cyclic rotation towards the left of its bit representation. The perfect shuffle permutation of a vector defined on a group $\mathcal{Z}_n$ is defined as the matrix S_n acting by

$$S_n\cdot x(q) = x(\sigma(q)). \tag{3}$$

Note 1. Permutation matrices defined in this way multiply together in a way opposite to what one may first think: if π_1 and π_2 are two permutations of the domain, and if the corresponding permutation matrices P_1 and P_2 are defined as in (3), we obtain

$$P_2\cdot P_1\cdot x(q) = P_1 x(\pi_2(q)) = x(\pi_1\circ\pi_2(q)),$$

where $\circ$ denotes function composition.

The Cooley–Tukey splitting of the FFT matrix is then given by the following lemma.

LEMMA 2.1. *Let S_n be the perfect shuffle. Then the Fourier matrix F_n can be factored as*

$$S_n\cdot F_n = (I_2\otimes F_{n/2})\cdot T_n\cdot(F_2\otimes I_{n/2}),$$

where T_n, the twiddle matrix, is diagonal. T_n acts on vectors in $\mathcal{G} = \mathcal{Z}_2 \otimes \mathcal{Z}_{n/2}$ according to

$$T_n \cdot x((g_1, g_0)) = \begin{cases} x((g_1, g_0)) & \text{if } g_1 = 0, \\ e^{2\pi i g_0/n} \cdot x((g_1, g_0)) & \text{if } g_1 = 1, \end{cases}$$

for all $g \in \mathcal{G}$, where (g_1, g_0) denotes the components of $g \in \mathcal{Z}_2 \otimes \mathcal{Z}_{n/2}$.

Proof. This is checked by a straightforward computation. For a survey of similar results, see, e.g., [24]. □

The Cooley–Tukey FFT algorithm applies this formula recursively. Starting with a matrix F_n, we first multiply from the left by S_n, then by $I_2 \otimes S_{n/2}$, $I_4 \otimes S_{n/4}$, and so on down to $I_{n/4} \otimes S_4$. This gives the complete binary factorization of F_n:

$$\begin{aligned} (4) \qquad & (I_{n/4} \otimes S_4) \times (I_{n/8} \otimes S_8) \cdots (I_2 \otimes S_{n/2}) \cdot S_n \times F_n \\ & = (I_{n/2} \otimes F_2) \times \left(I_{n/4} \otimes [T_4 \times (F_2 \otimes I_2)]\right) \cdots \left(I_2 \otimes \left[T_{n/2} \times (F_2 \otimes I_{n/4})\right]\right) \\ & \quad \times T_n \times (F_2 \otimes I_{n/2}). \end{aligned}$$

A multiplication of a vector by $I_{2^k} \otimes F_2 \otimes I_{2^l}$ is called a *butterfly* operation, and multiplication by $I_{2^k} \otimes T \otimes I_{2^l}$ a *twiddle* operation. The right-hand side in (4) is thus a series of butterfly and twiddle operations.

Let σ_l denote the perfect shuffle of the right-most l bits, i.e.,

$$\sigma_l(g) = (g_{k-1}, g_{k-2}, \ldots, g_l, g_{l-2}, g_{l-3}, \ldots, g_0, g_{l-1}).$$

Since $(I_{n/2^l} \otimes S_{2^l}) \cdot x(g) = x(\sigma_l(g))$, we find that

$$\begin{aligned} (I_{n/4} \otimes S_4) \cdot (I_{n/8} \otimes S_8) \cdots (I_2 \otimes S_{n/2}) \cdot S_n \cdot \, x(g) &= x(\sigma_{k-1} \circ \sigma_{k-2} \circ \cdots \circ \sigma_2(g)) \\ &= x((g_0, g_1, \ldots, g_{k-1})). \end{aligned}$$

Thus, as is well known, the results appear in *bit-reversed* order.

In the following we will describe how (4) is turned into a superparallel algorithm. This is best explained through pseudocodes.

The instruction stream of the superparallel FFT routine tells all the processors that they should compute the matrix–vector product as in the right-hand side of (4). It is, however, left to each processor to decide whether it should participate in each butterfly and twiddle operation, or if it should rest idle instead. Each processor must also decide which values it should use for the twiddle factors. By these decisions, (4) can be tuned to produce any matrix product of the form

$$(5) \qquad I_{n_l} \otimes F_{n_{l-1}} \otimes I_{n_{l-2}} \otimes F_{n_{l-3}} \otimes \cdots \otimes F_{n_1} \otimes I_{n_0},$$

where n_i are powers of two. We will see later that it is very useful to be able to include identity matrices in the product as well.

When the superparallel FFT routine is initially called, each processor must, of course, know which product it wants to compute, and this information must be *consistent.* For example, if a processor wants to participate in a butterfly operation, then its corresponding neighbor must want the same.

The type of matrix product each processor wants to compute is stored *locally* in an *active_list* containing *active groups*, i.e., a list of beginning bits and end bits for each term F_{n_i} in (5).

Example 1. Suppose the address space of the computer is 12 bits. If a processor wants to participate in the product

$$I_{2^3} \otimes F_{2^4} \otimes I_{2^2} \otimes F_{2^3},$$

then it creates the following list, containing two active groups

$$\text{active\_list} = \left\{ \begin{matrix} (8,5) \\ (2,0) \end{matrix} \right\},$$

indicating that F_{2^4} affects bits 8 to 5 and F_{2^3} affects bits 2 to 0. (It is most natural to start from the left since the reduction starts with the left-most bit first.)

See §3 for some practical examples.

Given the active list of each processor, the pseudocode in Algorithm 1 describes the main part of the algorithm. First, we present some preliminaries.

1. We assume that there are 2^k processors in the network and altogether 2^k data points. Initially, each processor stores the data point $x(g)$, where $g = (g_{k-1}, \dots, g_0)$ is the binary address of the processor. In §2.3 we discuss the case where there are more data points.

2. The symbol $\oplus$ denotes *exclusive or* (xor), i.e.,

$$\begin{aligned} 0 \oplus 0 &= 0, \\ 0 \oplus 1 &= 1, \\ 1 \oplus 0 &= 1, \\ 1 \oplus 1 &= 0. \end{aligned}$$

3. The *if*'s in the pseudocode denote parallel if's. That is, the processors that do not fulfill the condition in the test are masked as inactive and wait idle until the others are finished.

4. The final answers appear in bit-reversed order within each active group, e.g., in Example 1, processor $(g_{11}, g_{10}, \dots, g_0)$ will end up with the answer

$$\hat{x}((g_{11}, g_{10}, g_9, g_5, g_6, g_7, g_8, g_4, g_3, g_0, g_1, g_2)). \tag{6}$$

ALGORITHM 1.

```
# Basic version of superparallel FFT code
for $i = k-1, 0, -1$ # loop over all bits from left to right
    if ($i \in$ active_group) then
        # next line contains the interprocessor communication
        $tmp := x((g_{k-1}, \dots, g_{i+1}, g_i \oplus 1, g_{i-1}, \dots, g_0))$
        if ($g_i = 0$) then
            $x := x + tmp$
        else
            $x := tmp - x$
            $tw :=$ twiddle_factor($i$,active_group,$g$)
            $x := tw * x$
        endif
    endif
endfor
```

The algorithm takes $\mathcal{O}(p)$ parallel steps, where p is the number of bits that are active for some of the processors. For example, if the algorithm computes a collection of equally sized transforms, p is given as the $\log_2$ of the number of points in each transform.

From Lemma 2.1 we find that the twiddle factors are computed in the following ways.

ALGORITHM 2.

```
# Computation of twiddle factors
function twiddle_factor($i$, active_group,$g$)
    $ra$ := position_of_rightmost_bit_in_active_group
    # The right-hand side below is the binary representation of a
    real number
    $\theta = 0.g_{i-1}g_{i-2}\cdots g_{ra}$
    $tw := e^{2\pi\theta\sqrt{-1}}$
    return($tw$)
endfunction
```

It is amazing that such a simple code can simultaneously handle any collection of different-sized FFTs, even of different dimensionalities! The pseudocode is even simpler than the standard one-dimensional radix-2 FFT for a sequential computer. The simplicity of the code results from the inherent natural parallelism contained in the FFT.

2.3. Doubling the speed with bitswaps. The basic code has two flaws that should be corrected. First of all, it assumes that the total number of data points exactly matches the number of processors. The code must also be able to handle the case where more data points are available. For certain machines (e.g., the Connection Machine) the user is free to configure the number of *virtual processors*, i.e., the machine can, in a transparent way, pretend that it has more processors than it physically has. This is done in microcode. For other machines (e.g., the MasPar), the mapping of multiple points to a single processor is taken care of by either a high-level language compiler or, explicitly, by the programmer. We shall see that in the latter case it is possible to correct the second major drawback of the basic code, which is its efficiency.

Suppose the basic code is used to compute one large one-dimensional FFT, i.e., all the processors have a single active set covering all the address bits. First, all the processors swap one number with another processor; afterwards half of the processors compute a sum (while the rest are idle); and finally the second half of the processors compute a difference and perform the twiddle multiplication (while the first half is idle). Thus in a large part of the code, half of the machine is doing nothing.

Now assume that we have at least twice as many data points as processors. The way to handle this is to introduce some extra bits which refer to different locations in the memory of each processor. For example, if there are four times as many data points as processors, each processor stores four numbers and we get two extra address bits, which we call *memory bits.* Hereafter, we assume that the *left-most* bit is always a memory bit. The location of the other memory bits is of no importance, and, to simplify the exposition, we forget the rest of the memory bits and assume that we have *exactly* twice as many data points as we have processors. We define a *bitswap permutation* as a swapping of the left-most bit with another bit

$$B^i \cdot x((g_{k-1}, \ldots, g_i, \ldots,)) = x((g_i, \ldots, g_{k-1}, \ldots,)).$$

Note that B^i is its own inverse. The bitswap can be performed by the following pseudocode.

Algorithm 3.

```
# Computation of $x := B^i \cdot x$
function bitswap($i, x$)
   if $(g_{k-1} \neq g_i)$ then
      $x((g_{k-1}, \ldots, g_i, \ldots)) := x((g_{k-1} \oplus 1, \ldots, g_i \oplus 1, \ldots))$
   endif
endfunction
```

Note that the condition is true for exactly one of the two numbers in each processor, thus every processor is active and swaps exactly one number with another processor. The bitswap transfers all the "action" from bit i to the left-most bit by:

$$I_{n/2^i} \otimes [T_{2^i} \cdot (F_2 \otimes I_{2^{i-1}})] = B^i \cdot \tilde{T}_{2^i} \cdot (F_2 \otimes I_{n/2}) \cdot B^i, \tag{7}$$

where

$$\tilde{T}_{2^i} = B^i \cdot (I_{n/2^i} \otimes T_{2^i}) \cdot B^i$$

is a diagonal matrix of order n. Equation (7) can be substituted into (4). After some consideration, one realizes that the left-most bitswap in each substituted term commutes with everything to the left of it, and can be pulled out to the left and multiplied over to the other side of the equation. This gives the following "bitswap version" of (4):

$$\begin{aligned} &B^0 \cdot B^1 \cdots B^{k-1} \cdot (I_{n/4} \otimes S_4) \cdot (I_{n/8} \otimes S_8) \cdots (I_2 \otimes S_{n/2}) \cdot S_n \cdot F_n \\ &= \tilde{T}_2 \cdot (F_2 \otimes I_{n/2}) \cdot B^0 \cdot \tilde{T}_4 \cdot (F_2 \otimes I_{n/2}) \cdot B^1 \cdots \tilde{T}_{n/2} \cdot (F_2 \otimes I_{n/2}) \\ &\quad \times B^{k-2} \cdot \tilde{T}_n \cdot (F_2 \otimes I_{n/2}) \cdot B^{k-1}, \end{aligned} \tag{8}$$

where $\tilde{T}_2$ and B^{k-1} are identity matrices included to make the formula more symmetric.

In the basic variant of the algorithm, the data points were shuffled locally, i.e., they were only permuted to other processors with the same active_list as the processor where they started. In the bitswap version of the algorithm, data points are shuffled more globally, to processors that initially contained a different active_list. This means that in order to compute twiddle factors correctly, the active_list must also be bitswapped. Furthermore, if *any* processor wants to do a bitswap, we must force *all* processors to participate, even if they are not inside an active group. This is to ensure that the bitswaps define permutations of the data set. We use the word "all" to emphasize that no processor is allowed to be idle, and the word "any" to test if a parallel logical expression is true for at least one processor. This leads to the bitswap variant of the superparallel FFT.

Algorithm 4.

```
# Bitswap version of superparallel FFT code
for $i = k-1, 0, -1$ # loop over all bits from left to right
   if any ($i \in$ active_group) then
      all
         bitswap($i, x$)
         bitswap($i$,active_list)
      end all
   endif
   if ($i \in$ active_group) then
      $tmp := x((0, g_{k-2}, \ldots))$
```

```
            $x((0, g_{k-2}, \ldots)) := tmp + x((1, g_{k-2}, \ldots))$
            $x((1, g_{k-2}, \ldots)) := tmp - x((1, g_{k-2}, \ldots))$
            $tw :=$ twiddle_factor ($i$, active_group,g)
            $x((1, g_{k-2}, \ldots)) := tw * x((1, g_{k-2}, \ldots))$
        endif
    endfor
```

Note that the function twiddle_factor is the same as in the basic version of the algorithm.

In the computational part, this code computes two points in the same time that it takes the basic version to compute one point. If we neglect the bitswap of the active_list, each processor swaps one number for each i. Since each processor is computing twice as many numbers as in the basic version, we have also reduced the communication time by a factor of two. In practice, the code is divided into two parts, a symbolic preprocessing stage where the twiddle factors are computed, and a numerical phase where the actual transforms are done. In many situations the same geometries are used for many different transforms. Then the symbolic computation can be done once, and the cost of bitswapping the active_list and computing tw can be neglected. Thus, *on a* SIMD *machine, the bitswap version of the superparallel* FFT *is twice as fast as the basic version.*

The main drawback of the bitswap version is the more complicated scrambling of the results. As we shall see in §4, the bitswap version leaves the data in a much more disordered state than the basic version. For many applications, it is not necessary to unscramble the data. It suffices to know the identities of the results in each processor, and this is straightforward to compute. In other applications it is, however, desirable to unscramble the results. As we shall see in §4, efficient algorithms exist for unscrambling the results of both the basic and the bitswap version.

3. Defining problems and checking consistency.

Formally, the only restriction of the possible active_lists is that if a processor wants to participate in a multiplication by F_2, then its partner should want to do the same. However, this requirement alone allows some rather pathological cases, such as a two-dimensional field of numbers where every column wants to first do an FFT in the y-direction, but only the first column wants to afterwards do a transform in the x-direction as well. We cannot see any practical use for such a possibility. For debugging, it is useful to have the possibility of checking the active_lists for inconsistencies of this kind. In such situations, the unscrambling algorithms in the next section will give unpredictable results. We therefore want stricter rules for the possible active_lists. The following rule is general enough to allow all possible interesting transforms.

DEFINITION 1 (consistent active_lists). Given a point with active_list **al**, the set of bits not in any of the active groups is called *inactive*. The collection of all the active_lists is *consistent* if all points with the same values in the inactive bits also share the same active_list **al**.

It is a straightforward matter to check for consistency in this sense: simply mimic the FFT code, but instead of performing the butterfly and twiddle multiplications, check whether or not the active_lists of the neighbors are equal.

DEFINITION 2. A group of points sharing the same active_list and the same values for the inactive bits is called an FFT *chunk*. The values of the inactive bits define the *site* of the chunk.

The number of points in a chunk is given as 2^k, where k is the total number of *active* bits (bits inside the active groups).

We will now see some examples showing that the active_lists can easily be set up to solve rather complicated problems.

Example 2. Given a two-dimensional field of 512×256 points, suppose we want to divide the field into 8×8 square tiles, each of size 64×32, and perform two-dimensional FFTs on each tile. This is simply done by giving each point the active list

$$\text{active\_list} = \left\{ \begin{array}{c} (13,8) \\ (4,0) \end{array} \right\}$$

and calling the FFT routine.

This example also easily extends to the case where the domain is divided into rectangles of different sizes (as long as the sizes are powers of 2). For example, if we want to merge two tiles neighboring each other in the x-direction into one tile, we simply modify the active lists of these points to

$$\text{active\_list} = \left\{ \begin{array}{c} (13,8) \\ (5,0) \end{array} \right\}.$$

Example 3. Suppose we have a $64 \times 64 \times 64$ three-dimensional grid and we want to do an FFT on each two-dimensional $y-z$ plane. This is a practical problem arising from using spectral methods to simulate flow between plates. This problem is defined by giving each point the active list

$$\text{active\_list} = \left\{ \begin{array}{c} (17,12) \\ (11,6) \end{array} \right\}.$$

Compared to the more standard approach (for sequential computers) of solving these problems by looping over the independent FFTs, calling one-dimensional FFTs, transposing the data, looping and calling one-dimensional FFTs again, and finally transposing the matrix again, it is amazing how easily the superparallel FFT can be called to solve relatively complicated problems.

Now suppose we have a collection of different-sized problems, and that we are free to choose their initial positions as we like. We will show that it is always possible to solve these problems in an address space of size the smallest power of 2 larger than or equal to the sum of the sizes of all the subproblems. If there are no geometric relations between the different chunks, it is natural to store each chunk in a contiguous part of the address space. We are, however, not always allowed to pack the different chunks tightly together. Since the lowest address in a chunk is obtained by setting all the active bits to 0, the following *alignment requirement* must be satisfied.

LEMMA 3.1 (memory alignment). *An* FFT *chunk, with a total of k active bits, stored contiguously in the address space, must start in a memory location dividing* 2^k, *i.e., the right-most k bits are zero.*

If the alignment requirement is fulfilled, then the address succeeding the chunk must also divide 2^k. We conclude with the following theorem.

THEOREM 3.2. *If different-sized chunks are ordered in decreasing order according to their size, they can be stacked together contiguously with no interleave in the address space.*

COROLLARY 3.3. *Less than* 50 *percent of the address space is wasted due to the memory alignment requirement.*

Proof. The memory space needed is the smallest power of 2 containing all subproblems. □

Example 4. Suppose we have three one-dimensional FFTs of lengths 2, 4, and 8. Then we need four bits to define the problems:

```
chunk 1:
   site :   $q_3 = 0$
   active_list :  {(2,0)}
chunk 2:
   site :   $q_3 = 1$;  $q_2 = 0$
   active_list :  {(1,0)}
chunk 3:
   site :   $q_3 = 1$;  $q_2 = 1$;  $q_1 = 0$
   active_list :  {(0,0)}
```

The two points $(1,1,1,0)$ and $(1,1,1,1)$ are left unused.

4. Unscrambling the results. For many applications it is not necessary to unscramble the results. A call to an FFT is often followed with a call to an inverse FFT, and in between, the Fourier coefficients are often just multiplied by some diagonal matrix. If one has a pair of transforms, the FFT from ordered to scrambled, and the inverse working backwards from scrambled to unscrambled, then all one usually needs to know is the identities of each Fourier coefficient. After performing the (symbolic) FFT, this can easily be computed in parallel by computing backwards, using the information contained in the active_lists of the results and the information about which bits have been bitswapped.

Example 5. In a point $(g_{11}, g_{10}, \dots, g_0)$ the active_list is as in example (1) after the FFT (with bitswaps). Without bitswaps the point would contain the result as in (6). Suppose all bits have been swapped in the FFT; then the point contains the number

$$\begin{aligned} &B^{k-1} \cdots B^1 \cdot B^0 \cdot \hat{x}((g_{11}, g_{10}, g_9, g_5, g_6, g_7, g_8, g_4, g_3, g_0, g_1, g_2)) \\ &\quad = \hat{x}((g_2, g_{11}, g_{10}, g_9, g_5, g_6, g_7, g_8, g_4, g_3, g_0, g_1)). \end{aligned}$$

Note that a sequence of bitswaps produces an inverse perfect shuffle.

There are also, however, many applications where it is necessary to obtain the the results in unscrambled order (see, e.g., [3] and [16]). In this section we will derive efficient unscrambling algorithms. To study permutation algorithms it is necessary to assume a model for the computer interconnection network. A particularly useful network is the Beneš network [2], [13], [19]. This is one of the simplest networks capable of performing any permutation of 2^k objects. A permutation algorithm for a Beneš network can easily be transformed into efficient permutation algorithms for a variety of different networks (see the comments below, and for more details, [17]).

A (masked) *bit-exchange* permutation of bit i is given by

$$x((q_{k-1}, q_{k-2}, \dots, q_i, \dots, q_0)) \to x((q_{k-1}, q_{k-2}, \dots, q_i \oplus \chi, \dots, q_0)),$$

where $\chi(q) \in \{0,1\}$ is a Boolean function. We require this to be a permutation. This is equivalent to the condition

$$\chi((q_{k-1}, q_{k-2}, \dots, q_i, \dots, q_0)) = \chi((q_{k-1}, q_{k-2}, \dots, q_i \oplus 1, \dots, q_0)),$$

i.e., χ is a function independent of the value of bit q_i. The Beneš network on 2^k data points performs a series of $2k-1$ bit exchanges, where the bits are exchanged in the fixed order

$$q_{k-1}, q_{k-2}, \ldots, q_1, q_0, q_1, \ldots, q_{k-2}, q_{k-1}.$$

By choosing the functions χ_i (i.e., setting the switches in the network) in the right manner, it is known that the Beneš network can perform any permutation of its 2^k inputs. The major problem is the computation of the functions χ_i. This is called the *B-setting* problem. Given a B-setting for a permutation, it is straightforward to solve permutation problems for a variety of different interconnection topologies (see [17] for details).

• *The "many points per processor problem."* Given a network with $p = 2^d$ processors, $d < k$, where there are 2^m, $m = k - d$, data points per processor, suppose we know how to perform a general permutation of data spread by one point per processor. For example, on the MasPar the cheapest way to perform one-point-per-processor permutations is to call a "router" routine (black box permutation algorithm). The problem is: *How few router calls are necessary to perform a given permutation of the complete data set?* If the address bits are ordered so that the left-most m bits are memory bits, we see that the sequence of exchanges

$$q_{k-1}, \ldots, q_{k-m}$$

only involves a local reordering of the data. The sequence

$$q_{k-m-1}, \ldots, q_1, q_0, q_1, \ldots, q_{k-m-1}$$

can be collapsed into one permutation and performed by 2^m calls to the router (one for each memory location). Finally the sequence

$$q_{k-m}, \ldots, q_{k-1}$$

is again only a local reordering. Thus a B-setting provides us with an algorithm that can perform a general permutation by only 2^m router calls. For many permutations it is impossible to do this with fewer calls.

• *Shuffle-exchange networks* [25] can directly perform the same permutations as a Beneš network, by running it forward k steps and backward $k-1$ steps. Alternatively, the B-settings can be transformed into algorithms for performing the permutation in $3k-1$ forward steps [19], or by a somewhat more complex algorithm, in $3k-3$ forward steps [9].

• *Butterfly and omega networks* are equivalent to k forward loops in a shuffle-exchange network. Thus the comments above apply to computers based on these networks.

• *Hypercubes* are extensions of Beneš networks, where *any* of the k bits can be exchanged in each step. Thus a hypercube network can emulate a Beneš network simply by exchanging the bits in a fixed order. This will, however, only use one out of the possible k wires extending from each processor in each step. The main problem is how to efficiently use the full bandwidth of the network. In the case of 2^m points per processor, the technique above splits the permutation task into 2^m independent tasks. The full bandwidth of the network can be utilized by running k of these tasks in parallel over different wires. This can be done by cyclically shifting the order of the bits in each independent permutation, i.e., k different permutations are done simultaneously by exchanging the bits in the following order:

Task 0:	q_{k-1}	q_{k-2}	$\cdots$	q_1	q_0	q_1	$\cdots$	q_{k-1}
Task 1:	q_{k-2}	q_{k-3}	$\cdots$	q_0	q_{k-1}	q_0	$\cdots$	q_{k-2}
Task 2:	q_{k-3}	q_{k-4}	$\cdots$	q_{k-1}	q_{k-2}	q_{k-1}	$\cdots$	q_{k-3}
$\vdots$								
Task $k-1$:	q_0	q_{k-1}	$\cdots$	q_2	q_1	q_2	$\cdots$	q_0

If there is only one point per processor, this technique may be used, provided that each point is associated with so much data that it can be split into k different parts. Each part can then be permuted on a different set of wires.

• *Two-dimensional mesh-connected computers.* There are several ways to emulate a Beneš network on a two-dimensional mesh. These schemes involve $\mathcal{O}(N^{1/2})$ time for performing permutations. See [12] for special tricks related to the MasPar hardware.

• *Multistage crossbar networks,* where each crossbar performs a general permutation of 2^s wires, can be programmed by collapsing s consecutive bit exchanges into general permutations of 2^s points. For example, the hardware underlying the MasPar "router" construct is a three-stage crossbar. A B-setting of a permutation can, in principle, be used to increase the speed of this kind of hardware. However, we have not tried this approach for the MasPar, since there is no high-level language access to programming the router hardware.

Let $N = 2^k$. The best known algorithms for solving the general B-setting problem takes $\mathcal{O}(N\log(N))$ time on a sequential computer, $\mathcal{O}(N^{1/2})$ time on a mesh-connected parallel computer with $N^{1/2} \times N^{1/2}$ processors, $\mathcal{O}(k\log^3(N))$ time on a hypercube-connected computer with $\mathcal{O}(N^{1+1/k})$ processors where $1 \leq k \leq \log(N)$, and $\mathcal{O}(\log^2(N))$ time on an N-processor shared-memory computer [19]. Thus, unless the FFTs are to be performed many times with the same geometries, the general B-setting algorithms will be too expensive compared to the time of running the numerical parts of the FFTs. The high cost of the general B-setting algorithms has led to a search for classes of permutations that can be B-set *efficiently* on a parallel computer (i.e., where solving the B-setting problem takes no more time than actually performing the permutation). See [17], [13], [19], and [10] for fast B-setting algorithms. Unscrambling the results of the basic superparallel FFT (Algorithm 1) belongs to the classes of permutations that can be solved by the algorithms in these papers. For the more difficult problem of unscrambling the results of the bitswap version (Algorithm 4) these algorithms will generally fail, and a new algorithm is needed.

Before attacking this general unscrambling problem, we study two simpler problems:

1. Finding B-settings for unscrambling the results of the basic version of the superparallel FFT.

2. Finding B-settings for a product of bitswap permutations.

We define the shorthand notation

$$q_i := q_i \oplus \chi$$

to denote the permutation

$$x((\ldots, q_i, \ldots)) \to x((\ldots, q_i \oplus \chi, \ldots))\ .$$

A commonly used basic permutation is the xor of two different bits

$$q_i := q_i \oplus q_j.$$

As long as $i \neq j$, this defines a permutation. (When $i = j$ it is *not* a permutation, since $q_i \oplus q_i = 0$—two different numbers are collapsed into a single memory address.)

Let $\bar{q}_i$ denote the original values of the bits. A bit reversal, e.g.,

$$x((q_3, q_2, q_1, q_0)) \to x((q_0, q_1, q_2, q_3)),$$

can be performed by the following B-setting:

ALGORITHM 5. UNSCRAMBLING BIT-REVERSED DATA (example with four bits).

$$\begin{aligned}
q_3 &:= q_3 \oplus q_0 = \bar{q}_3 \oplus \bar{q}_0 \\
q_2 &:= q_2 \oplus q_1 = \bar{q}_2 \oplus \bar{q}_1 \\
q_1 &:= q_1 \oplus q_2 = \bar{q}_1 \oplus \bar{q}_2 \oplus \bar{q}_1 = \bar{q}_2 \\
q_0 &:= q_0 \oplus q_3 = \bar{q}_0 \oplus \bar{q}_3 \oplus \bar{q}_0 = \bar{q}_3 \\
q_1 &:= q_1 = \bar{q}_2 \\
q_2 &:= q_2 \oplus q_1 = \bar{q}_2 \oplus \bar{q}_1 \oplus \bar{q}_2 = \bar{q}_1 \\
q_3 &:= q_3 \oplus q_0 = \bar{q}_3 \oplus \bar{q}_0 \oplus \bar{q}_3 = \bar{q}_0
\end{aligned}$$

Note that when there is an odd number of bits, there will be a bit in the middle that is left unchanged. This algorithm extends readily to the more general case where several active groups are to be bit reversed. Even the situation where there are several different FFT chunks with different active groups to be bit reversed can be handled by this method, since each chunk is to be unscrambled within its own site, and since the above method only modifies the active bits of each chunk, i.e., there is no interference between different chunks. Thus this process solves the unscrambling problem for the basic algorithm.

Now to the problem of finding a B-setting for a product of bitswaps: For simplicity, we assume that all the bits are to be bit-swapped. (If some of the bits should not be bit-swapped, we simply ignore them and proceed as below.)

As noted in Example 5, a product of bitswaps amounts to doing an inverse perfect shuffle. Let

$$\bigoplus_{i=0}^{k-1} q_i = q_0 \oplus q_1 \oplus \cdots \oplus q_{k-1};$$

then

$$q_j := \bigoplus_{i=0}^{k-1} q_i \tag{9}$$

is a permutation. By performing this permutation cyclically, one obtains the perfect shuffle.

ALGORITHM 6. PERFECT SHUFFLE PERMUTATION.

$$q_{k-1} := \bigoplus_{i=0}^{k-1} q_i = \bigoplus_{i=0}^{k-1} \bar{q}_i$$
$$q_{k-2} := \bigoplus_{i=0}^{k-1} q_i = \bigoplus_{i=0}^{k-2} \bar{q}_i \oplus \bigoplus_{i=0}^{k-1} \bar{q}_i = \bar{q}_{k-1}$$
$$q_{k-3} := \bigoplus_{i=0}^{k-1} q_i = \bigoplus_{i=0}^{k-3} \bar{q}_i \oplus \bigoplus_{i=0}^{k-1} \bar{q}_i \oplus \bar{q}_{k-1} = \bar{q}_{k-2}$$
$$\vdots$$
$$q_j := \bigoplus_{i=0}^{k-1} q_i = \dots = \bar{q}_{j+1}$$
$$\vdots$$
$$q_1 := \bigoplus_{i=0}^{k-1} q_i = \bar{q}_2$$
$$q_0 := \bigoplus_{i=0}^{k-1} q_i = \bar{q}_1$$

on return only q_{k-1} needs correction :

$$q_{k-1} := \bigoplus_{i=0}^{k-1} q_i = \bar{q}_0$$

Now we are prepared to study the general problem of unscrambling the results of the bitswap version of the superparallel FFT. The permutation to be performed is a product of a local bit reversal of each active group (performed differently for each chunk), and a global inverse perfect shuffle of all the chunks. In a given chunk, there are some bits that can be exchanged without changing the site of the chunk. There are other bits that are more critical, since modifying them will change the site of the chunk. We call these bits *site defining.* A modification of the site-defining bits must be coordinated with all the other chunks where the bits are site defining. Note that the site-defining bits may change during the course of the algorithm. At some stage even a linear combination of all the bits in a chunk may be site defining.

Consider an example where the address space consists of five bits. Given a chunk with $q_4 = 1$, $q_0 = 0$, and active group $(3,1)$, Table 1 shows the site-defining bits during an inverse perfect shuffle.

TABLE 1

Step	Permutation	Site (after permutation)	
0	none	$q_4 = 1;$	$q_0 = 0$
1	$q_4 := \bigoplus_{i=0}^{4} q_i$	$\bigoplus_{i=0}^{4} q_i = 1;$	$q_0 = 0$
2	$q_3 := \bigoplus_{i=0}^{4} q_i = \bar{q}_4$	$q_3 = 1;$	$q_0 = 0$
3	$q_2 := \bigoplus_{i=0}^{4} q_i = \bar{q}_3$	$q_3 = 1;$	$q_0 = 0$
4	$q_1 := \bigoplus_{i=0}^{4} q_i = \bar{q}_2$	$q_3 = 1;$	$q_0 = 0$
5	$q_0 := \bigoplus_{i=0}^{4} q_i = \bar{q}_1$	$q_3 = 1;$	$\bigoplus_{i=0}^{4} q_i = 0$
6	$q_4 := \bigoplus_{i=0}^{4} q_i = \bar{q}_0$	$q_3 = 1;$	$q_4 = 0$

Steps 1, 2, 5, and 6 are critical, and since they change the site, they must be coordinated with the other chunks. Let the motion defined in (9) be denoted the *collective motion.* The idea of the unscrambling algorithm is to force all the chunks to follow the collective motion when they are in a critical section. Outside critical sections we impose a local motion within a chunk in such a manner that the active groups finally become bit reversed.

There is a critical section two bits long when entering an active group (the last bit outside, and the first bit inside, the active group), and one two bits long when exiting (the two bits succeeding the active group). Thus the movement in the two bits following the active group is forced according to the collective motion. The main problem is to arrange the local movement in the active group such that the two succeeding bits come out correctly when they participate in the collective motion.

The following algorithm solves this problem; thus it solves the problem of unscrambling the bitswap version of the superparallel FFT. For proof of correctness of this algorithm, see [15].

ALGORITHM 7. UNSCRAMBLING OF BITSWAP FFT RESULTS (local motion passing an active group (r, s)). The chunk wants to perform the motion:

$$(\ldots, q_{r+1}, q_r, q_{r-1}, \ldots, \ldots, q_s, q_{s-1}, \ldots, \ldots)$$
$$\downarrow$$
$$(\ldots, \ldots, q_{r+1}, q_s, q_{s+1}, \ldots, q_{r-1}, q_r, q_{s-1}, \ldots)$$

1. The bits q_{r+1}, q_r, q_{s-1} and q_{s-2} are critical sections. They follow the collective motion:

$$q_j := \bigoplus_{i=0}^{k-1} q_i.$$

2. The bits $q_{r-1}, q_{r-2}, \ldots, q_s$ are to be bit reversed. This is done as in Algorithm 5, with the following modifications:
   - On the way "down," the bits q_{r-1} and q_s are exchanged according to:

$$\begin{aligned} q_{r-1} &:= \textstyle\bigoplus_{i=0}^{k-1} q_i \oplus q_s, \\ q_s &:= \textstyle\bigoplus_{i=0}^{k-1} q_i \oplus q_{r-1}. \end{aligned}$$

   - On return, the bits q_s and q_{r-1} are corrected to their final values according to:

$$\begin{aligned} q_s &:= \textstyle\bigoplus_{i=s-1}^{p} q_i \quad \text{where } p = \lfloor (r+s)/2 \rfloor - 1, \\ q_{r-1} &:= q_{r-1} \oplus q_{s-1}. \end{aligned}$$

The computation of B-settings for the unscrambling of bitswap FFT results takes $\mathcal{O}(p)$ parallel steps, where p is the number of bits that are active for some of the processors.

We illustrate the algorithm with an example.

Example 6. $r = 9; s = 2$. The symmetry point is $p = 5$. The motion of this chunk passing the active group is done as:

$$\begin{aligned} q_9 &:= \textstyle\bigoplus_{i=0}^{k-1} q_i = \bar{q}_{10} \\ q_8 &:= \textstyle\bigoplus_{i=0}^{k-1} q_i \oplus q_2 = \bar{q}_2 \oplus \bar{q}_9 \\ q_7 &:= q_7 \oplus q_3 = \bar{q}_3 \oplus \bar{q}_7 \\ q_6 &:= q_6 \oplus q_4 = \bar{q}_4 \oplus \bar{q}_6 \\ q_5 &:= q_5 = \bar{q}_5 \\ q_4 &:= q_4 \oplus q_6 = \bar{q}_6 \\ q_3 &:= q_3 \oplus q_7 = \bar{q}_7 \end{aligned}$$

$$q_2 := \bigoplus_{i=0}^{k-1} q_i \oplus q_8 = \bar{q}_8 \oplus \bar{q}_6 \oplus \bar{q}_7 \oplus \bar{q}_9$$
$$q_1 := \bigoplus_{i=0}^{k-1} q_i = \bar{q}_9$$
$$q_0 := \bigoplus_{i=0}^{k-1} q_i = \bar{q}_1$$

on return the bits are corrected as

$$q_2 := q_1 \oplus q_2 \oplus q_3 \oplus q_4 = \bar{q}_8$$
$$q_6 := q_6 \oplus q_4 = \bar{q}_4$$
$$q_7 := q_7 \oplus q_3 = \bar{q}_3$$
$$q_8 := q_8 \oplus q_1 = \bar{q}_2$$

In this example $r - s$ is an odd integer, and the bit q_5 is left invariant. When $r - s$ is even the algorithm is similar, but without an invariant bit in the middle.

5. Implementations on MasPar. MasPar is a SIMD machine with up to 16,384 processors arranged in a two-dimensional array, with toroidal wraparound on the edges. The machine used in this study is a $64 \times 128 = 8192$-processor machine at the University of Bergen, Norway. There are essentially three different ways of permuting a data set spread by one element per processor:

1. The xnet. This is a mechanism for sending the data set a given distance d in one of the eight directions: north, northwest, west, and so on. There are no restrictions on the distance d. This mechanism is very fast for short distances, but the time grows proportionally with d; thus long xnets are costly.

2. Piped xnet. This is a fast version of xnet, where the time is (almost) independent on d. It can, however, only be used for sparse data sets, where all the processors between the senders and the receivers are idle. It is very useful for computations of inner products and log sums, but not for our FFTs.

3. The router. This is a general (black box) construct for performing arbitrary permutations of the data set (spread by one element per processor). The underlying hardware is a three-stage crossbar switch. The time for a router call depends on the permutation, but is comparable to an xnet of the longest distance (64). For general permutations it is definitely the cheapest mechanism.

A more detailed general description of the MasPar MP-1 computer can be found in [4], [5], and [20].

The MasPar implementation of the superparallel FFTs is based on the bitswap version of the algorithm. For more details on its use and performance, see [18]. The code is divided into three parts: a symbolic preprocessing phase, the actual transform phase, and the scrambling/unscrambling phase. The symbolic phase computes the twiddle factors and schedules the unscrambling of the results. If the user only wants the results in scrambled order, it computes the necessary pointers for referencing the Fourier coefficients. The symbolic phase needs to be called only once for each configuration of geometries, and the same data is used both for the forward and the inverse FFTs.

In Table 2, we show the elapsed time for various combinations of transforms having combined lengths of 16,384 and 262,144, using 32-bit arithmetic. For simplicity, all transforms have the same length, but we emphasize that any combination of transforms can be processed. These timings *exclude* the time for the symbolic preprocessing. The present version of the symbolic part of the code is not optimized, and uses between 5 and 10 times as much time as the actual transforms. There is, however, substantial room for improvement of this part of the code.

TABLE 2
Time in milliseconds and computational speed for FFT *algorithm* (8192-*processor* MP-1).

Length	#Transforms	Time (ms) without unscrambling	Mflops without unscrambling	Time (ms) with unscrambling	Mflops with unscrambling
Tot. no. of points: 16384					
8	2048	1.1	229	2.2	108
64	256	2.5	198	3.5	139
512	32	4.0	185	5.4	137
8192	2	5.4	199	7.2	147
16384	1	6.1	187	8.1	141
Tot. no. of points: 262144					
8	32768	14.5	270	33.8	117
8192	32	84	203	105	162
262144	1	124	191	175	135

The speed of transforms without unscrambling is, to a moderate degree, dependent on the geometries; when a lot of small transforms are processed, the communication distances are shorter and the speed somewhat higher than for long transforms. The cost of the unscrambling is dominated by the cost of calling the router. This is done once for every set of 8192 points, and takes between 0.5 and 1 millisecond per call (with some special tricks employed to increase the speed). Thus the cost of unscrambling grows linearly in N. This explains the phenomenon that, whereas short transforms are faster than long without unscrambling, long transforms are faster than short with unscrambling.

Acknowledgments. The author would like to thank Professor Petter Bjørstad for suggesting improvements to an early version of the paper and the CERFACS researchers for their hospitality and support.

REFERENCES

[1] D.H. BAILEY, FFTs *in external or hierarchical memory*, J. Supercomput., 4 (1990), pp. 23–35.

[2] V.E. BENEŠ, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[3] P. BJØRSTAD, J. COOK, H. MUNTHE-KAAS, AND T. SØREVIK, *Implementation of a* SAR *processing algorithm on MasPar* MP-1208, Rep. No. 57, Dept. of Informatics, Univ. of Bergen, Norway, 1991.

[4] T. BLANK, *The Mas Par* MP-1 *Architecture*, Proc. IEEE Compcon, Spring 1990, IEEE, Feb. 1990.

[5] P. CHRISTY, *Software to support massively parallel computing on the MasPar* MP-1, Proc. of IEEE Compcon, Spring 1990, IEEE, Feb. 1990.

[6] J.W. COOLEY AND J.W. TUKEY, *An algorithm for machine calculations of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[7] A. GRAHAM, *Kroneker Products and Matrix Calculus: With Applications*, John Wiley, New York, 1981.

[8] D.W. HILLIS AND G.L. STEELE, JR., *Data Parallel Algorithms*, Comm. ACM, 29 (1986), pp. 1170–1189.

[9] S.-T. HUANG AND S.K. TRIPATHI, *Finite state model and compatibility theory: New analysis tools for permutation networks*, IEEE Trans. Comput., C-35 (1986), pp. 591–601.

[10] ———, *Self-routing technique in perfect-shuffle networks using control tags*, IEEE Trans. Comput., 37 (1988), pp. 251–256.

[11] S.L. JOHNSSON, M. JACQUEMIN, AND C.-T. HO, *High radix* FFT *on Boolean cube networks*, Tech. Rep. NA89-7, Thinking Machines Corporation, Cambridge, MA, 1989.

[12] C.L. KUSZMAUL, FFT *communications requirement optimizations on massively parallel architectures with local and global interprocessor communications capabilities*, MasPar report

TW009.0790, presented at International Society for Optical Engineering, July 8–13, San Diego, CA, 1990.

[13] J. LENFANT, *Parallel Permutations of Data: A Beneš network control algorithm for frequently used permutations*, IEEE Trans. Comput., C-27 (1978), pp. 637–647.

[14] H. MUNTHE-KAAS, *Topics in linear algebra for vector and parallel computers*, Ph.D. thesis, Department of Mathematical Sciences, Univ. of Trondheim, Norway, 1989.

[15] ———, *Super Parallel* FFT*s*, Rep. No. 52, Dept. of Informatics, Univ. of Bergen, Norway, 1991.

[16] ———, *Super Parallel* FFT*s with applications to seismic processing*, Proc. IEEE Workshop on Parallel Architechtures for Seismic Data Processing, Glasgow, Scotland, 1991.

[17] ———, *Practical parallel permutation procedures*, Dept. of Informatics, Univ. of Bergen, Norway, 1991.

[18] ———, *MasPar* SPFFT *users guide*, Dept. of Informatics, Univ. of Bergen, Norway, 1992.

[19] D. NASSIMI AND S. SAHNI, *A self-routing Beneš network and parallel permutation algorithms*, IEEE Trans. Comput., C-30 (1981), pp. 332–340.

[20] J. NICKOLLS, *The design of the MasPar* MP-1*, A cost effective massively parallel computer*, Proc. IEEE Compcon, Spring 1990, IEEE, Feb. 1990.

[21] S.D. PARKER, *Notes on shuffle/exchange-type switching networks*, IEEE Trans. Comput., C-29 (1980), pp. 213–222.

[22] D. PARKINSON, *Super parallel algorithms*, in Supercomputing, NATO ASI series F, Vol. 62, Springer-Verlag, Berlin, New York, 1989.

[23] M.C. PEASE, *The indirect binary n-cube microprocessor array*, IEEE Trans. Comput., C-26 (1977), pp. 548–573.

[24] D.J. ROSE, *Matrix identities of the fast Fourier transform*, Linear Algebra Appl., 29 (1980), pp. 423–443.

[25] H.S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-20 (1971), pp. 153–161.

[26] P.N. SCHWARZTRAUBER, FFT *algorithms for vector computers*, Parallel Comput., 1(1984), pp. 45–63.

[27] ———, *Multiprocessor* FFTs, Parallel Comput., 5(1987), pp. 197–210.

[28] C. WU AND T. FENG, *The universality of the shuffle-exchange network*, IEEE Trans. Comput., C-30 (1981), pp. 324–331.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 368–388, March 1993

NUMERICAL EXPERIENCE WITH A CLASS OF ALGORITHMS FOR NONLINEAR OPTIMIZATION USING INEXACT FUNCTION AND GRADIENT INFORMATION*

RICHARD G. CARTER†

Abstract. For optimization problems associated with engineering design, parameter estimation, image reconstruction, and other optimization/simulation applications, low accuracy function and gradient values are frequently much less expensive to obtain than high accuracy values. The computational performance of trust region methods for nonlinear optimization is investigated for cases when high accuracy evaluations are unavailable or prohibitively expensive, and earlier theoretical predictions that such methods are convergent even with relative gradient errors of 0.5 or more is confirmed. The proper choice of the amount of accuracy to use in function and gradient evaluations can result in orders-of-magnitude savings in computational cost.

Key words. unconstrained optimization, trust region methods, inexact function evaluations, inexact gradients, variable-accuracy simulations

AMS(MOS) subject classifications. 65K05, 65K10, 49D37, 65C20

1. Introduction.

Consider the nonlinear optimization problem

$$\underset{x \in \Re^n}{\text{minimize}} \quad f(x), \tag{1}$$

where the function f has gradient ∇f, with ∇f assumed to be Lipschitz continuous. We are concerned with numerically solving this problem when function and gradient values are not known exactly.

Problems of this nature frequently occur in engineering design, parameter estimation, and many other situations. Consider, for example, the design of a heat sink for transferring excess heat away from an electronic component. Given the geometry of the sink (expressed, perhaps, as the spacing, thickness, and length of each cooling fin) and the heat flux from the component, we can mathematically model the temperature distribution in the sink and the surrounding medium by a system of partial differential equations (PDEs). If we wish to find the design geometry that minimizes the time-averaged temperature of the component, we must numerically solve this system of PDEs at each iteration of the optimization algorithm to determine an approximate value for the objective function f. Furthermore, a value for the gradient of f must be computed at each iteration through directly solving a different system of PDEs (sensitivity or adjoint equations [18]), through successively perturbing each component of x and recomputing f to obtain a finite difference approximation, or through applying an automatic differentiation package to the computations used to approximate f. Clearly, exact function and gradient values are not attainable for such a problem, and the computational expense of any approximation at a given iteration increases very rapidly as the required accuracy is increased. Let h be the discretization mesh size selected and m_d the number of spatial dimensions in the PDE, so that the number of

* Received by the editors October 30, 1989; accepted for publication (in revised form) March 25, 1992.

† Stoner Associates, Inc., 5177 Richmond Avenue, Suite 1075, Houston, Texas 77056-6736. This research was supported by National Aeronautics and Space Administration contracts NAS1-18107 and NAS1-18605 while the author was at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23665. This work was also 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, and by Army Research Office grant DAAL03-89-C-0038.

elements in the discretization is given by $k = O((1/h)^{m_d})$. If the order of the solution method is given by error $= O(h^{m_o})$, and the computational expense for the linear algebra associated with the problem is CPU $= O(k^{m_l})$, we have

$$\text{CPU/iteration} = O\left(\text{error}^{-l}\right) \tag{2}$$

for $l = m_d m_l / m_0$. Although this estimate is admittedly crude, it seems to hold for many applications and indicates that computational expense per iteration can rise *extremely* rapidly as more accurate solutions are required. In our example, if a two-dimensional model of the sink is being used with a direct linear algebra solver and an $O(h^2)$ solution method, we would have $m_d = 2, m_l = 3, m_0 = 2$; hence l would be 3 and a thousand-fold increase in computational time would be needed to increase the accuracy of any given approximation by one digit. Methods using iterative linear algebra solvers and problems involving systems of ordinary differential equations (ODEs) rather than PDEs tend to be more benign with much smaller values of l ($\frac{1}{4}$ to $\frac{1}{10}$ for most ODE applications), but computational expense still increases geometrically with accuracy. Similar results often hold for stochastic, particle-based simulation methods.

Since low accuracy function and gradient evaluations can be orders of magnitude less expensive than high accuracy evaluations, it behooves us to consider optimization algorithms that do not require the maximum possible accuracy at each iteration. Trust region methods are a natural candidate for investigation because of their reputation for robustness and efficiency. A number of authors have established global convergence results for trust region methods using inexact gradients [13], [4], [19], and inexact function evaluations have also been treated [3]. In this paper, we investigate the numerical behavior of the algorithms presented in [4] and [3] in order to answer the following questions.

1. How much error can we allow in our evaluations before the algorithm fails? Does this level agree with theoretical predictions?
2. The performance of the algorithm will certainly decrease when less accurate evaluations are used. How fast does performance degrade and how problem-dependent is the rate of degradation?
3. How does this lessened performance balance with the greatly decreased computational cost associated with less accurate evaluations?
4. How well do the techniques suggested in [3] for estimating and controlling gradient error work in practice?

The remainder of this paper is organized as follows.

In §2, we present the trust region algorithm and review the conditions on permissible levels of error established in [3]. These conditions depend on some of the parameters of the trust region method but are remarkably relaxed: typically, relative errors in the gradient of 0.5 or more are permissible. In §3, we examine the performance of the algorithm on the set of standard test problems from Moré, Garbow, and Hillstrom [14] when synthetically generated errors are added to the gradients at each iteration. Our results confirm the theoretical predictions for the algorithm, and we note that the number of iterations required by the algorithm tends to increase exponentially with the relative error induced in the gradient. When balanced against (2), however, we find that allowing low accuracy evaluations is still attractive. In §4, we examine the performance of the algorithm on a multidimensional numerical integration problem in order to confirm our results without resorting to synthetically induced errors or invoking (2). Both direct and Monte Carlo variable-accuracy integration techniques are demonstrated in this example. In §5 we solve a parameter identification problem found in the literature and verify a technique for estimating and controlling error when the gradient is approximated by finite differences. Section 6 summarizes our results.

2. Trust region algorithms and permissible error in function and gradient evaluations.

The class of trust region methods we consider for solving (1) each generate a sequence of iterates$\{x_k\}$ by approximately solving a sequence of constrained quadratic model problems. Each local quadratic model is of the form

$$\Psi_k(x_k + s) = f_k + g_k^T s + \tfrac{1}{2}s^T B_k s \tag{3}$$

where f_k is an approximation to $f(x_k)$, g_k is an approximation to the gradient $\nabla f(x_k)$, and the symmetric matrix $B_k \in \Re^{n\times n}$ is an approximation to the Hessian matrix $\nabla^2 f(x_k)$. At each iteration, we take $x_{k+1} = x_k + s_k$, where s_k is an approximate solution to the *trust region subproblem*

$$\underset{s\in\Re^n}{\text{minimize}}\, \Psi_k(x_k + s) \quad \text{subject to } \|D_k s\| \le \Delta_k. \tag{4}$$

The positive scalar Δ_k is known as the *trust radius*, and the nonsingular matrix $D_k \in \Re^N$ is the *scaling* or *preconditioning* matrix (often taken to be a fixed diagonal matrix). At each iteration, Δ_k is adjusted so that the ball $\|D_k s_k\| \le \Delta_k$ represents the region over which we expect (3) to adequately model the function f.

A number of techniques are available for computing an approximate solution to (4). An excellent survey of the main classes of these methods can be found in [13]. In most of our computations, we chose to use an *optimal locally constrained* (OLC) technique [7]. Similarly, a number of techniques can be used to generate the Hessian approximation $\{B_k\}$, but we selected the popular BFGS secant update

$$B_{k+1} = B_k + \frac{(g_{k+1} - g_k)(g_{k+1} - g_k)^T}{(g_{k+1} - g_k)^T s_k} - \frac{B_k s_k s_k^T B_k^T}{s_k^T B_k s_k}, \tag{5}$$

provided that

$$(g_{k+1} - g_k)^T s_k \ge 10^{-6}(g_{k+1} - g_k)^T(g_{k+1} - g_k), \tag{6}$$

and $B_{k+1} = B_k$ otherwise. Since g_k is only an approximation to $\nabla f(x_k)$, [4] and [3] suggest that upper bounds of the form

$$\|B_k\| \le c_1 \tag{7}$$

or

$$g_k^T B_k g_k \le c_1 g_k^T g_k \tag{8}$$

be directly enforced for some appropriately large c_1. This could be easily done by using the replacement operation

$$B_k := \min\left\{1, \frac{c_1}{\|B_k\|}\right\} B_k \tag{9}$$

at each iteration. Although bounds such as (7) and (8) are needed to establish convergence results, in practice we found (9) unnecessary.

A simple version demonstrating the salient features of the trust region approach is as follows.

ALGORITHM 1. An implementation of the trust region method.
Let the constants $0 < \eta_1 < \eta_2 < \eta_3 < 1$ be prespecified. Select an initial guess $x_0 \in \Re^N$ and an initial trust radius Δ_0. Compute f_0 and g_0, and compute or initialize B_0.

```
For $k = 0, 1, \ldots$ until "convergence" do:
    (a) Determine an approximate solution $s_k$ to problem (4).
    (b) Calculate the predicted function reduction
```

$$\text{pred}_k(s_k) = -g_k^T s_k - \tfrac{1}{2} s_k^T B_k s_k \tag{10}$$

```
        and the computed function reduction
```

$$\text{cred}_k(s_k) = f_k - f_{k+1}. \tag{11}$$

```
        If necessary, recompute $f_{k+1}$ and/or $f_k$ to greater accuracy.
    (c) Compute the ratio
```

$$\rho_k = \frac{\text{cred}_k(s_k)}{\text{pred}_k(s_k)}. \tag{12}$$

```
    (d) If $\rho_k < \eta_1$, then the step is unacceptable. Set $\Delta_k := \frac{1}{10}\Delta_k$ and return to (a).
    (e) If $\eta_1 \le \rho_k < \eta_2$, then set $\Delta_{k+1} = \frac{1}{2}\Delta_k$,
        Else if $\eta_3 < \rho_k \le (2 - \eta_3)$, then set $\Delta_{k+1} = 2\Delta_k$,
        Else set $\Delta_{k+1} = \Delta_k$.
    (f) Set $x_{k+1} = x_k + s_k$, compute $g_{k+1}$, and compute or update $B_{k+1}$.
End loop
End algorithm
```

Typical values for the step acceptance/trust radius update parameters are $\eta_1 = 0.001$, $\eta_2 = 0.1$, and $\eta_3 = 0.75$. Note that no step is accepted unless $\rho_k \ge \eta_1$, and that the trust radius is never reduced unless $\rho_k \le \eta_2$. Further note that g_k is only computed once per major iteration.

Two conditions are required of the approximate function values. Define

$$\text{ared}_k(s_k) = f(x_k) - f(x_k + s_k). \tag{13}$$

We then require

$$|\text{ared}_k(s_k) - \text{cred}_k(s_k)| \le \zeta_{f,1}\, \text{pred}_k(s_k) \tag{14}$$

and

$$|\text{ared}_k(s_k) - \text{cred}_k(s_k)| \le \zeta_{f,2}\, |\text{cred}_k(s_k)| \tag{15}$$

for some constants $\zeta_{f,1}$ and $\zeta_{f,2}$. A stronger variation of these conditions that is typically more practical is

$$|f_{k+1} - f(x_{k+1})| + |f_k - f(x_k)| \le \zeta_{f,1}\, \text{pred}_k(s_k) \tag{16}$$

and

$$|f_{k+1} - f(x_{k+1})| + |f_k - f(x_k)| \le \zeta_{f,2}\, |\text{cred}_k(s_k)|. \tag{17}$$

Assuming error estimates are available for $|f_k - f(x_k)|$, [3] suggests that (16) and (17) be enforced by using the following procedure in place of step (b) of Algorithm 1.

PROCEDURE 2.
Let $\alpha \in (0,1)$ be prespecified. Given x_k, s_k, Ψ_k, and an estimate for $|f_k - f(x_k)|$, do the following.

(a) Calculate $\text{pred}_k(s_k)$ and $\text{emax} = \zeta_{f,1}\text{pred}_k(s_k)$.
(b) If necessary, recompute f_k to greater accuracy so that $|f_k - f(x_k)| \leq \alpha\,\text{emax}$.
(c) Compute f_{k+1} so that $|f_{k+1} - f(x_{k+1})| \leq (1-\alpha)\text{emax}$.
(d) Compute $\text{cred}_k(s_k)$. If (17) is satisfied, then exit procedure, else reduce emax and return to (b).

End procedure

The permissible level of error in the gradient evaluations can be characterized in two different ways. The preferable condition is

$$\frac{\|D_k^{-1}e_k\|}{\|D_k^{-1}g_k\|} \leq \zeta_g \tag{18}$$

for some constant ζ_g, with

$$e_k = g_k - \nabla f(x_k). \tag{19}$$

Under appropriate assumptions on f and $\{D_k\}$, (18) leads to the strong global convergence result $\lim_{k\to\infty}\|g_k\| = \lim_{k\to\infty}\|\nabla f(x_k)\| = 0$ [4, Thms. 3.3 and 4.1], [3, Thm. 3.4]. The weaker convergence result $\liminf_{k\to\infty}\|g_k\| = 0$ can be obtained [3, Thm. 3.3] using the condition

$$\frac{(D_k^{-1}e_k)^T\,(D_k^{-1}g_k)}{(D_k^{-1}g_k)^T\,(D_k^{-1}g_k)} \leq \zeta_g. \tag{20}$$

If each scaling matrix D_k is taken to be the identity, (18) and (20) become simply

$$\frac{\|e_k\|}{\|g_k\|} \leq \zeta_g \tag{21}$$

and

$$\frac{e_k^T g_k}{g_k^T g_k} \leq \zeta_g. \tag{22}$$

Finally, we must specify the allowed values for the error bounds $\zeta_{f,1}, \zeta_{f,2}$, and ζ_g. These values are given by the inequalities

$$\zeta_g + \zeta_{f,1} < 1 - \eta_2 \tag{23}$$

and

$$0 \leq \zeta_{f,2} < 1, \tag{24}$$

with $\zeta_g \geq 0$ and $\zeta_{f,1} \geq 0$. These limits are *remarkably* generous. If η_2 (the parameter controlling trust radius reduction) is 0.1, we could select $\zeta_{f,1} = 0.05, \zeta_{f,2} = 0.99$, and $\zeta_g = 0.8$—less than one significant bit of accuracy in the components of the gradient approximation.

3. Algorithm performance on Moré–Garbow–Hillstrom test problems with synthetically induced errors. The Moré–Garbow–Hillstrom test set [14] contains eighteen "typical" optimization problems. These problems are all algebraically defined, and thus function and gradient values are both inexpensive to compute and available to high accuracy. In order to test the effects of gradient error, we synthetically induced a random error into each gradient computed. More specifically, we computed a vector w_k with each component selected from a uniform distribution on [-1,1] and then set $g_k = \nabla f(x_k) + e_k$ with

$$e_k = 100 * w_k * \|\nabla f(x_k)\|/2^m, \tag{25}$$

where m is the smallest positive integer for which (21) is satisfied. For our first set of tests no errors were induced in the computation of function values.

For our optimization code, we used an implementation of the Dennis–Schnabel routines [5] using an "optimal locally constrained" step computation procedure (see [5, pp. 134–139] for details). The model Hessians B_k were computed using the standard BFGS procedure (5) with the normal test to ensure positive definiteness (6). We emphasize that no special techniques were used to account for gradient error when computing B_k, nor were safeguards such as (7), (8), or (9) enforced. The value for η_2 in the optimization code was 0.1, so that the theory in [4] and [3] suggests an upper limit of 0.9 for ζ_g. The constants η_1 and η_3 were selected to be 0.001 and 0.8, respectively. The Dennis–Schnabel "relative gradient" stopping criterion was used:

$$\max_{1\le i\le n} \frac{|(\nabla f(x_k))_i \max\{(x_k)_i, 1\}|}{\max\{|f_k|, 1\}} \le \epsilon_m^{1/3} \tag{26}$$

(where ϵ_m is machine epsilon and $(y)_i$ denotes the ith component of a vector y). The "true" gradient is used in (26) rather than the approximate gradient g_k to eliminate the possibility that e_k might also affect the convergence tests in our trials. A fairly stringent x convergence test was also used:

$$\|\tilde{s}_k\|_\infty \le \epsilon_m^{2/3}, \tag{27}$$

where $(\tilde{s}_k)_i \equiv |(s_k)_i|/\max\{(x_k)_i, 1\}$.

A run was also flagged as a failure if more than eight subiterations were attempted at any iteration without finding an acceptable new iterate.

For each problem in the test set, we considered 20 different values of ζ_g ranging from 0.0 to 0.95. For each value of ζ_g, we ran the algorithm with a number of different initializations for the random number generator used to compute e_k values, and tabulated the minimum, median, and maximum number of iterations required to converge to a local minimizer.[1] In all, over 5000 test cases were run on a network of SUN 3/50 workstations in FORTRAN 77 double precision.

For the majority of our problems, the failure test was rarely triggered until error levels reached 80 percent, and did not become prevalent until error levels reached 90 percent.[2] This corresponds well with the value predicted by theory.

Figures 1–5 show plots of our results for selected problems. Let $K(\zeta_g)$ denote the number of iterations required for convergence (using stopping criterion (26) and (27)) as a function of relative gradient error. The vertical axis in each plot represents

[1] Depending on the computational expense associated with a problem and the observed variability of results, between 5 and 100 test cases were run for each value of ζ_g.

[2] An exception was the Beale function, which began to trigger the failure test at 50 percent error.

the natural log of $K(\zeta_g)$, while the three traces in each plot represent the observed minimum, median, and maximum values.

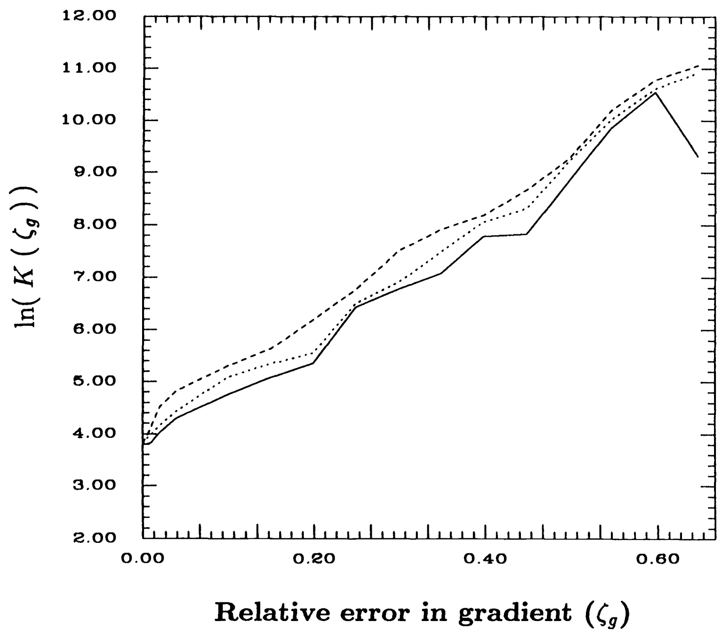


FIG. 1. *Number of iterations versus ζ_g for the Watson test function.*

Figures 1–3 are typical of most of our test cases, in which performance degrades exponentially:

$$K(\zeta_g) \approx K(0)\exp(b\zeta_g), \tag{28}$$

with observed decay coefficients b ranging from roughly 2 to 6 for the various problems. Figures 4 and 5 represent anomalous cases where $\ln(K(\zeta_g))$ has significant variation from linearity at small and large values of ζ_g.

Although the exponential performance degradation given by (28) is a telling argument against using low accuracy gradients if high accuracy evaluations are obtainable without greatly increased computational expense, low accuracy evaluations are still attractive in cases where the computational expense increases rapidly with increasing accuracy. Suppose, for instance, that

$$\text{CPU/iteration} \approx c_2\zeta_g^{-l} \tag{29}$$

for some constants c_2 and l. The total computational expense for solving a given problem will then be proportional to $\zeta_g^{-l}K(\zeta_g)$. Figures 6–10 show the predicted total computational cost of the median curves for $K(\zeta_g)$ in Figs. 1–5 for the values $l = \frac{1}{2}, 1$, and 2 (with each curve normalized so that the minimum value is 1.0).

Note that each curve of total computational cost increases very rapidly as $\zeta_g \to 0$ or $\zeta_g \to 1$, and has a relatively large, flat minima. This behavior holds for both the "typical" cases (Figs. 6–8) and the anomalous cases (Figs. 9 and 10). Interestingly,

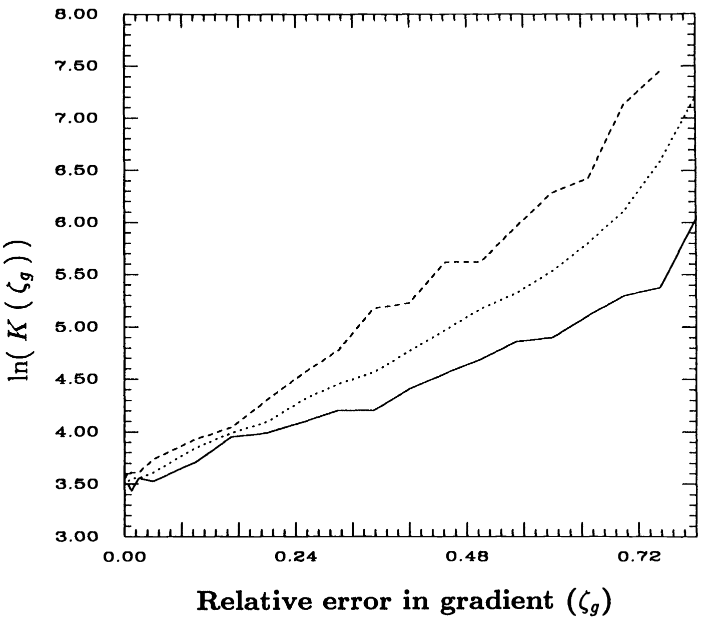


FIG. 2. *Number of iterations versus ζ_g for the Brown and Dennis test function.*

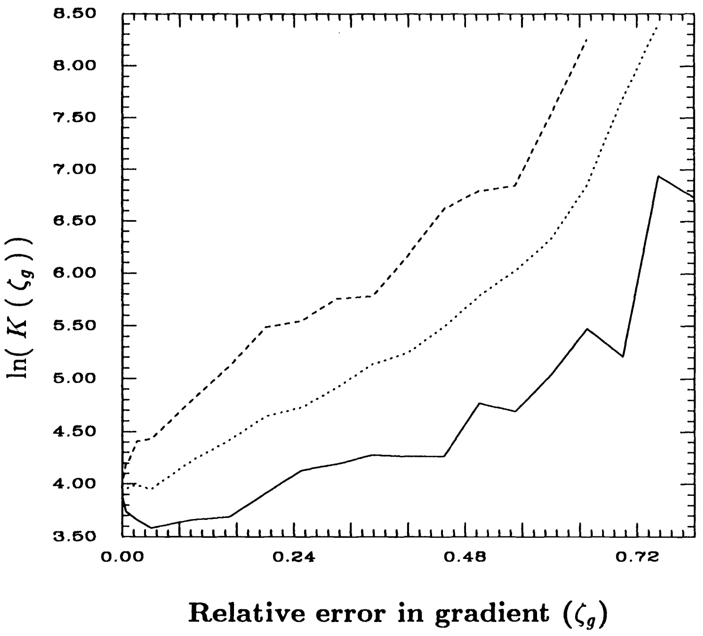


FIG. 3. *Number of iterations versus ζ_g for the extended Powell singular test function.*

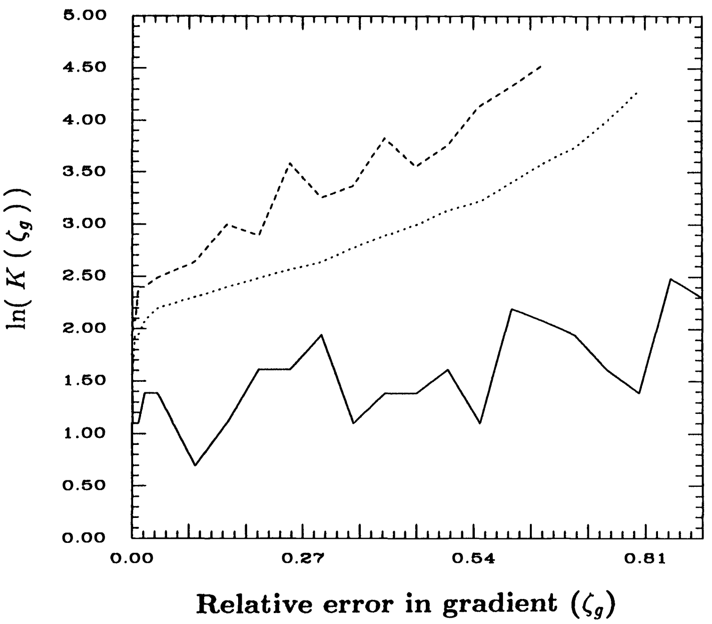


FIG. 4. *Number of iterations versus ζ_g for the Gaussian test function.*

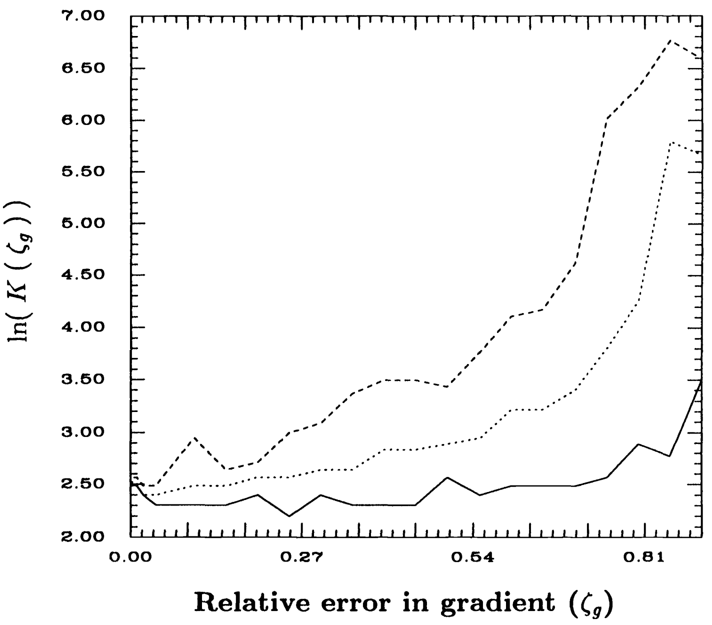


FIG. 5. *Number of iterations versus ζ_g for the trigonometric test function.*

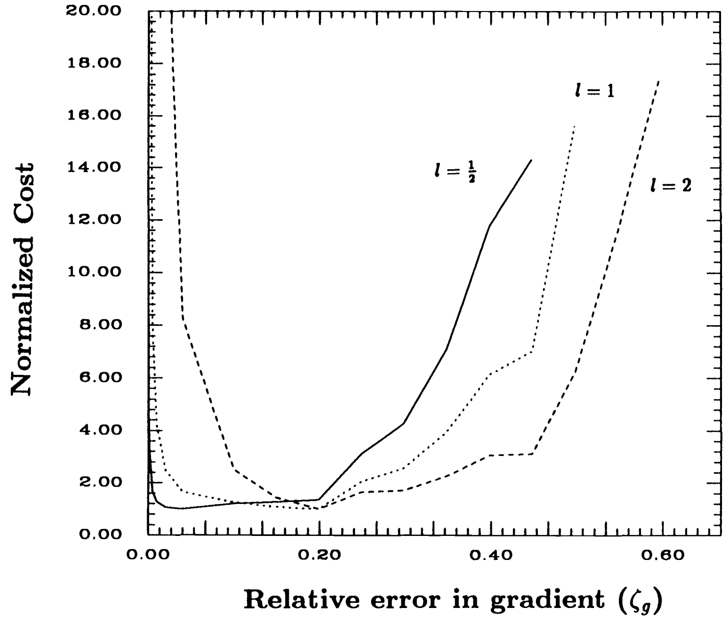


FIG. 6. *Computational cost profiles for the Watson test function.*

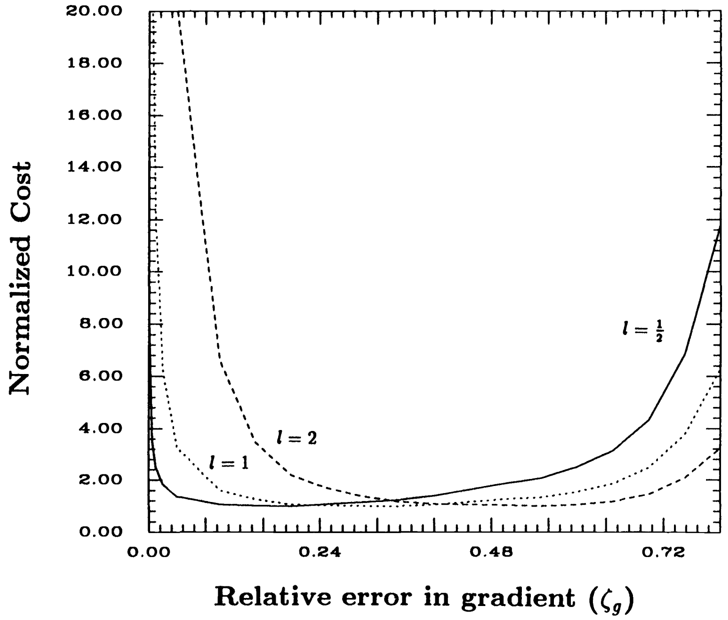


FIG. 7. *Computational cost profiles for the Brown and Dennis test function.*

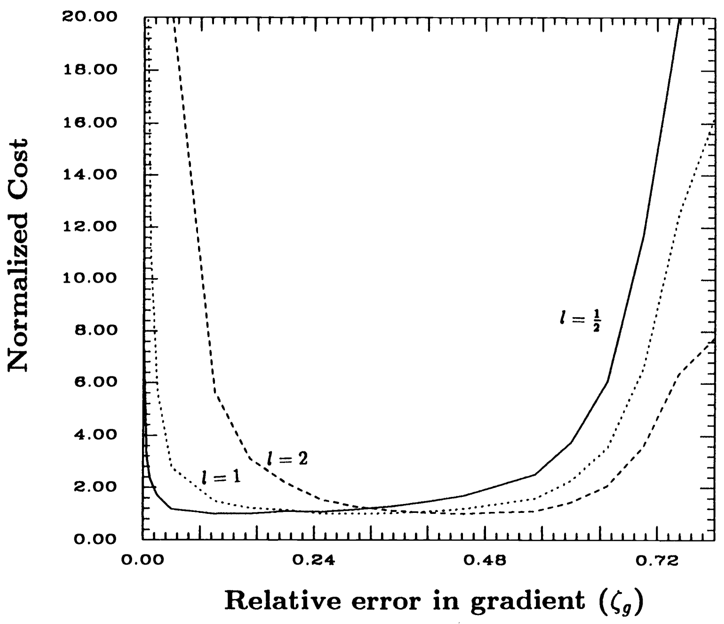


FIG. 8. *Computational cost profiles for the extended Powell singular test function.*

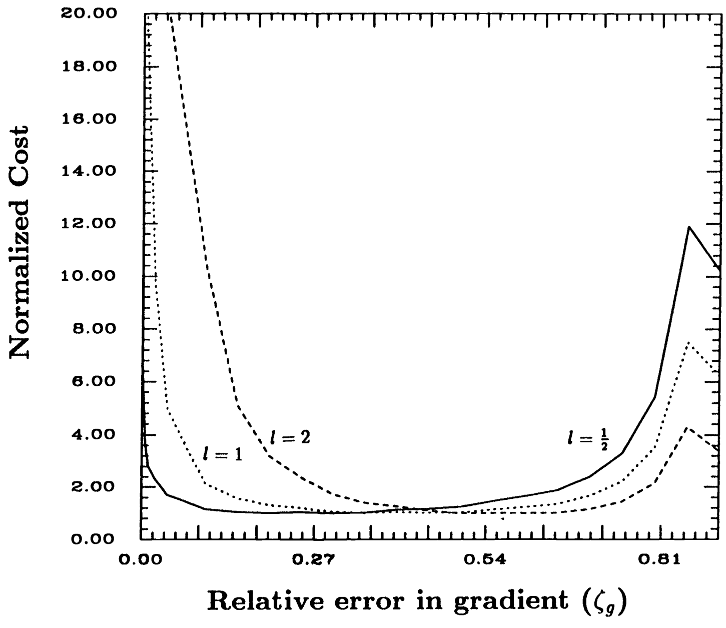


FIG. 9. *Computational cost profiles for the Gaussian test function.*

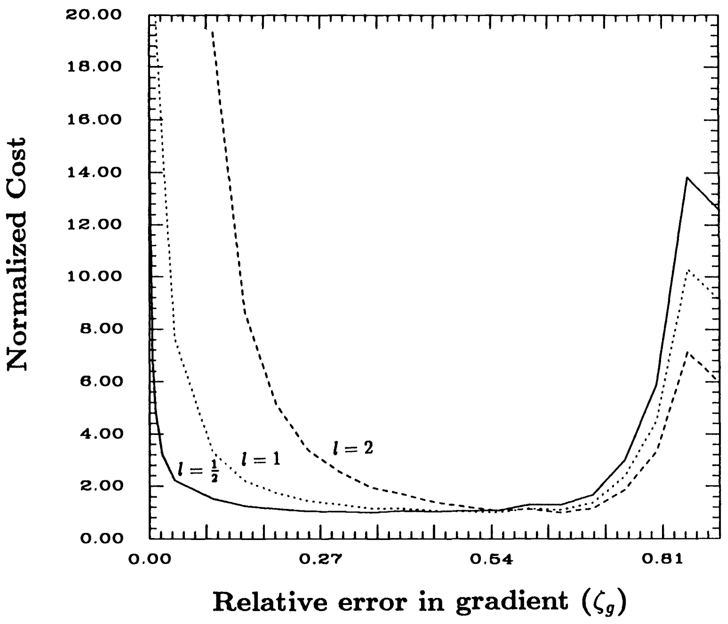


FIG. 10. *Computational cost profiles for the trigonometric test function.*

combining the idealizations (28) and (29) into

$$\text{CPU} \approx K(0)c_2\zeta_g^{-l}\exp(b\zeta_g) \tag{30}$$

yields the theoretical "best" value

$$\zeta_g^* = \frac{l}{b}. \tag{31}$$

Using the observed "typical" value $b = 4$ yields the rule-of-thumb choice

$$\zeta_g^* = \frac{l}{4}, \tag{32}$$

which worked quite well for all of our test problems.

For values of $\zeta_g \leq 0.75$, the x convergence test (27) was rarely triggered. As ζ_g approached $1 - \eta_2$, an increasing ratio of the trials terminated on (27) rather than the relative gradient test (26). These "solutions" are somewhat suspect, so iteration count data in the preceding figures at the highest error levels (0.75 up) should be considered an underestimate.

A number of variations on these numerical tests were also tried. Rather than (29), we also considered the idealization

$$\text{CPU/iteration} \approx c_2\zeta_g^{-l} + c_3, \tag{33}$$

where c_3 represents a fixed "overhead" cost per iteration. For moderate values of c_3, the character of the overall computational cost behavior remained unchanged. Errors in function values were also considered with ζ_g fixed at 0.1. The algorithm proved to be quite insensitive to these errors for $\zeta_{f,1} < 0.5$. Indeed, the algorithm works in practice even if $\zeta_{f,1} > 1$ provided the *average* value of $|\text{ared}_k(s_k) - \text{cred}_k(s_k)| \ / \ \text{pred}_k(s_k)$ is sufficiently less than $1 - \eta_2 - \zeta_g$. As pointed out in [3], this is a very reasonable result since function values are only used to update the trust radii, and a mistake at any given iteration will not cause the algorithm to fail.

4. Algorithm performance on problems involving numerical integration in several dimensions. The test problems of the previous section are well known and widely used by optimization researchers and software developers, and because they are algebraically defined, we were able to run an enormous number of test cases to examine the ranges of possible behavior in the presence of errors. It should be remembered, however, that our interpretation of these results rests both on the character of the synthetic noise added to each gradient evaluation and on idealization (29). In order to verify our results, we also tested our algorithm on several other problems.

Let $(y)_i$ denote the ith component of a vector y, and define

$$h_1(t) = \exp\left(-\sum_{i=1}^{n} \frac{(t)_i}{i}\right) \quad \text{and} \quad h_2(t,x) = \cos\left(\sum_{i=1}^{n} \frac{(t)_i}{(x)_i}\right). \tag{34}$$

Our problem is to find x to match the surfaces h_1 and h_2 over the box $\Omega = \{t : 0 \leq (t)_i \leq 1, i = 1, \ldots, n\}$ in the least squares sense. That is, we wish to minimize

$$f(x) = \int_{t\in\Omega} (h_2(t,x) - h_1(t))^2 dt. \tag{35}$$

The components of the gradient of f are simply

$$\frac{\partial f}{\partial (x)_k} = \int_{t\in\Omega} 2\ (h_2(t,x) - h_1(t)) \sin\left(\sum_{i=1}^{n} \frac{(t)_i}{(x)_i}\right) \frac{(t)_k}{(x)_k^2}\, dt. \tag{36}$$

For the purpose of demonstration, we will assume that n is small and calculate f and ∇f separately by numerical integrations using (a) a recursive Simpson's rule with uniform mesh,[3] and (b) a simple stochastic scheme. The starting point $x = [1\ 1 \ldots 1]^T$ is used in all of our runs, which were executed on an IBM RS-6000 in double precision FORTRAN 77.

In practice, values for f (and similarly the components of ∇f) can be computed to any desired accuracy by successively decreasing the stepsize h in Simpson's rule. For instance, halving the stepsize for sufficiently small h will result in decreasing the error by a factor of roughly 16. Thus we can approximate $|f_k^h - f(x_k)|$ by $|f_k^h - f_k^{h/2}|$ and $|f_k^{h/2} - f(x_k)|$ by $|f_k^h - f_k^{n/2}|/2^4$, where f_k^h is our computation for $f(x_k)$ using stepsize h. More specifically, for a given commanded accuracy ϵ, and two trial stepsizes h_k and h_{k-1}, we compute our next trial h by

$$h_{k+1} = \left\{\frac{0.9\epsilon}{|f_k^{h_k} - f_k^{h_{k-1}}|}\right\}^{1/4} \tag{37}$$

[3] In practice one would of course consider more sophisticated schemes such as Gaussian quadrature or adaptive stepsize methods (or simply solve the problem analytically), but for demonstration we selected the simplest possibility.

(and safeguarding h_{k+1} such that $\frac{1}{2}h_k \le h_{k+1} \le \frac{7}{8}h_k$). This simple rule works very well in practice for target error levels less than 0.1. Typically, the rule results in successively halving h until the estimated error is within an order of magnitude of the target error ϵ, and then selecting an "almost optimal" h in $[\frac{1}{2}h_k, \frac{7}{8}h_k]$. For $n > 2$, the total computational cost is little greater than the expense of evaluating f with the final h. We considered $n = 2$, $n = 3$, and $n = 4$. As n increases, the curve describing computational expense as a function of accuracy becomes increasingly hostile: CPU per f evaluation is $O(\epsilon^{-n/4})$.

For comparison, we first attempted to solve this problem naively requiring fixed absolute accuracy in f (and fixed relative accuracy in the components of g) of six digits, with $n = 4$. Using the same code as in the previous section and terminating when the predicted reduction became less than the specified accuracy in f, we reduced f from 0.71484 to 0.0263596 using 11,004 seconds of CPU time (just over three hours). Using seven digits gave the same function reduction in 16,619 seconds. In contrast, if we solved the same problem using a fixed accuracy of $\zeta_g = 0.1$ in the components of g and computing f values via Procedure 2 (with $\alpha = 0.5$, $\zeta_{f,1} = 0.4$, and $\zeta_{f,2} = 0.99$), we were able to solve the problem to the same accuracy in only 59 seconds—more than 200 times faster.

Clearly, a dramatic difference in computational expense can be expected in cases where computational expenses rise rapidly with increasing accuracy requirements. Much less dramatic effects are to be expected when the rise of computational costs is not so steep. Table 1 shows our results for a variety of values of n.

TABLE 1
Expense of solving (35) *for different gradient accuracies. Function and gradient values approximated separately by recursive Simpson's rule.*

	$n = 2$		$n = 3$		$n = 4$	
Commanded gradient accuracy	Final f value	CPU time (sec)	Final f value	CPU time (sec)	Final f value	CPU time (sec)
.1000000	.02734732	.67	.02840882	32	.02635958	59
.0500000	.02734732	.75	.02840882	41	.02635958	201
.0200000	.02734732	.64	.02840882	48	.02635958	250
.0100000	.02734732	.69	.02840882	43	.02635957	2347
.0050000	.02734732	.57	.02840882	53	.02635957	3560
.0020000	.02734732	.58	.02840882	56	.02635957	4185
.0010000	.02734732	.59	.02840882	48	.02635957	3745
.0005000	.02734732	.61	.02840882	67	.02635957	5200
.0002000	.02734732	.74	.02840882	70	.02635957	4370
.0001000	.02734732	.77	.02840882	71	.02635957	4458
.0000100	.02734732	.88	.02840882	77	.02635957	7694
.0000010	.02734732	1.12	.02840882	108	.02635957	10433
.0000001	.02734732	1.27	.02840882	134	.02635957	12716

If n is not small, stochastic integration techniques are, of course, preferable to recursive integration rules, since accuracy increases proportionally to the square root of the number of sample points independent of n: CPU per f evaluation is $O(\epsilon^{-2})$. We considered the simplest possible stochastic integration scheme: for a given x, compute a sequence of h_1 and h_2 values at a sequence of points $\{t_j\}$ uniformly distributed over Ω, take the mean of $\{(h_2(t_i, x) - h_1(t_i))^2\}$, $i = 1, \ldots, j$ to be the jth approximation to the integral, and estimate the error in our approximation by three standard deviations of the mean of $\{(h_2(t_i, x) - h_1(t_i))^2\}$. The components of g are computed by a similar integration. Our results are presented in Table 2.

The final row of Table 2 represents the maximum accuracy attempted for this approach; this run was terminated prematurely due to excessive time requirements (greater than 200 hours on the RS-6000). Note that the final function reduction attained for $\zeta_g = .002$ is actually *less* than that attained using $\zeta_g = .2$, yet $\zeta_g = .002$ uses almost 3812 times as much CPU time—truly a compelling argument for using the minimal sufficient accuracy on problems where costs rise this fast with increased commanded accuracy.

TABLE 2
Expense of solving (35) *for different gradient accuracies. Function and gradient values approximated separately by Monte Carlo integration.*

Commanded gradient accuracy	$n = 4$	
	Final f value	CPU time (sec)
.500	.03130167	98
.200	.02969897	168
.100	.02664009	2139
.050	.02717885	31466
.010	.02986443	37559
.005	.03036428	68837
.002	.03060412	640425
.001	<.03049246	>723550

5. Algorithm performance on a parameter identification problem. Our final example is a parameter identification problem from [11] and [12]. Consider the accidental release of the radioactive gas tritium into an enclosure surrounding a nuclear reactor. The tritium will react with water vapor in the containment to produce other tritium-based species via

$$T_2 + H_2O \rightleftharpoons HT + HTO, \tag{38}$$

and some of the HTO may be adsorped into the surface of the containment. This adsorped tritium species represents a significant cleanup problem.

Given the reaction rate constant in (38) and the adsorption and release rates of HTO on the surfaces of the containment, the physical problem can be modeled by a system of four coupled, initial-value ODEs:

$$Y'(t;x) = h(t, Y(t;x)), \qquad Y(0;x) = Y_0, \tag{39}$$

with the components $Y : \Re \times \Re^3 \to \Re^4$ being species concentrations. Unfortunately, the rate constants are not directly measurable. Maroni, Land, and Minkoff performed an experiment in which a known amount of tritium was introduced into a small enclosure and the total tritium concentration $(Y)_1 + (Y)_2 + (Y)_3$ was measured at m discrete time points. The rate constants $(x)_1, (x)_2,$ and $(x)_3$ can then be estimated by minimizing $f : \Re^4 \to \Re$ with

$$f(x) = \frac{1}{2}\sum_{i=1}^{m}\left[\sum_{j=1}^{3}(Y)_j(t_i, x) - O_i + (x)_4\right]^2 / (O_i)^2, \tag{40}$$

where O_i is the observed experimental concentration

$$O_i = Y_1(t_i) + Y_2(t_i) + Y_3(t_i), \tag{41}$$

and $(x)_4$ is an additional variable representing an unknown experimental bias in the instrument for measuring (41).

Equation (40) is a classical inverse problem. Note that each function evaluation for a given iterate x_k involves the numerical solution of four coupled ODEs. Gradient values can be computed via finite differences, or by the numerical solution of a larger system of coupled ODEs derived using the sensitivity [12] or adjoint equations (see, for instance, [18]). Although the latter techniques are usually preferable in practice, we used the more difficult approach of estimating g_k by finite differences so that we could investigate techniques suggested in [3] for estimating and controlling gradient error.

Our numerical experiments with (40) were designed as follows. In order to approximate f at a given point x_k, (39) was solved using ODEPACK [8], which uses an adaptive solution technique. An important feature of ODEPACK is that it allows a desired level of accuracy (either absolute or relative) to be prespecified for each component of Y. In order to achieve an accuracy of, say, ε_k in $f(x_k)$, we specified a desired accuracy of $\alpha_{k-1}\varepsilon_k$ in each component of Y, where α_{k-1} was the amount of accuracy lost due to cancellation in evaluating (40) at the last iteration. This simple procedure worked remarkably well: the actual error in f_k was typically in the range $[1/10\varepsilon_k, 2\varepsilon_k]$ in our preliminary tests of this technique.

Each gradient was initially approximated by a central difference formula using $2n$ extra function evaluations, where each function evaluation was computed with a specified desired relative accuracy of $\bar{\varepsilon}_k$. Denote this approximation $\bar{g}_k$. We then computed a more accurate estimate of the directional derivative of f in the direction $\bar{g}_k$ (or $(D_k^T D_k)^{-1}\bar{g}_k$ if the scaling matrix is not the identity) as suggested in [3], using the formula

$$\bar{d}_k = \frac{1}{2\delta_k}(f(x_k + \delta_k \bar{g}_k) - f(x_k - \delta_k \bar{g}_k)). \tag{42}$$

Each function evaluation in (42) was computed with a specified desired relative accuracy of $\bar{\varepsilon}_k/10$. The perturbation length δ_k was taken to be

$$\delta_k = \left(\frac{\bar{\varepsilon}_k}{10}\right)^{1/3} |f_k|/g_k^T g_k\ , \tag{43}$$

a value expected to perturb two-thirds of the accurate digits of f. Using (19) and (42), we can then estimate the error term in (22) via

$$\frac{e_k^T \bar{g}_k}{\bar{g}_k^T \bar{g}_k} = 1 - \frac{\nabla f(x_k)^T \bar{g}_k}{\bar{g}_k^T \bar{g}_k} \approx 1 - \frac{\bar{d}_k}{\|\bar{g}_k\|^2}. \tag{44}$$

If this error was larger than the desired gradient error level ζ_g, then $\bar{\varepsilon}_k$ was decreased before the next iteration; if it was significantly larger, then $\bar{\varepsilon}_k$ was immediately decreased and $\bar{g}_k$ was recomputed. (In practice, this seldom occurred except at the first gradient computation.) Figure 11 shows the agreement between actual and requested gradient error. Even with the simple procedures used to adjust $\bar{\varepsilon}_k$, the error estimate given by (42) and (44) allows us to control, with reasonable certainty, the level of accuracy in $\bar{g}_k$.

The approximation $\bar{g}_k$ can be further improved at no additional cost by setting

$$g_k = \frac{\bar{d}_k}{\|\bar{g}_k\|^2}\bar{g}_k \tag{45}$$

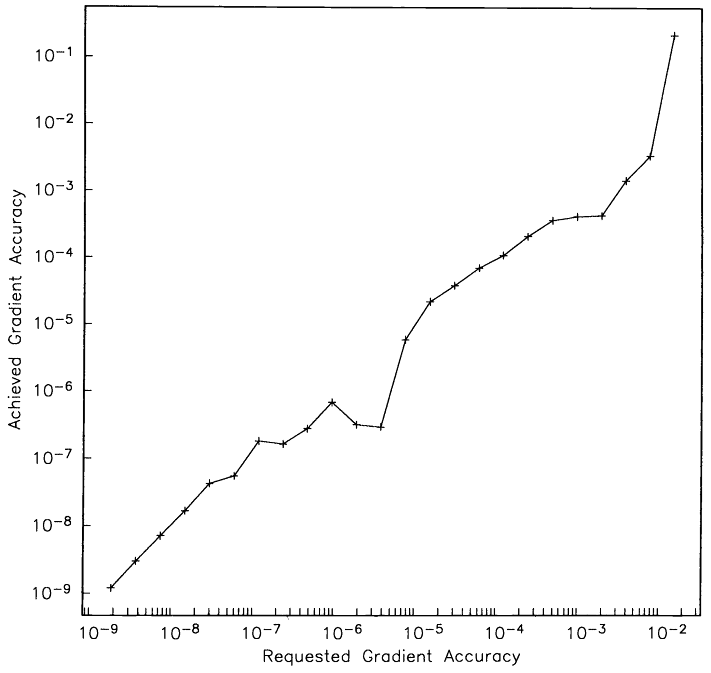


FIG. 11. *Actual versus commanded error in the gradient for the tritium parameter estimation problem.*

so that

$$\frac{e_k^T g_k}{g_k^T g_k} = 1 - \frac{\nabla f(x_k)^T \bar{g}_k}{\bar{d}_k}. \tag{46}$$

Figure 12 shows the CPU time required to achieve a given level of accuracy using (45) in addition to the previously discussed procedure for adjusting $\bar{\varepsilon}_k$. We see that computational expense increases geometrically as accuracy increases.[4]

Given these methods of evaluating f_k and g_k to some specified accuracy, we recorded the computational time required to solve (40) for a number of different values of ζ_g, and for the following three cases.

1. Each f_k value was computed to high relative accuracy (10^{-8}), and each g_k value was computed as previously described *including* the correction (45).

2. Each f_k was computed to high relative accuracy (10^{-8}), and each g_k was computed as described previously but *without* doing correction (45).

3. Each f_k was computed using Procedure 2 with $\zeta_{f,1} = 0.1$ and $\zeta_{f,2} = 0.99$, and each g_k value was computed as previously described *including* correction (45).

[4] The idealized cost profile (29) yields a very close fit to this plot if l is taken to be $\frac{1}{10}$. However, tests with different values of x_k showed that with better empirical form this problem is

$$\text{CPU/gradient evaluation} \approx c_2 \|g_k\|^{-1} \zeta_g^{-1/10}.$$

Nonetheless, the rule-of-thumb choice (32) with $l = \frac{1}{10}$ proved to be close to the optimal selection in our numerical tests.

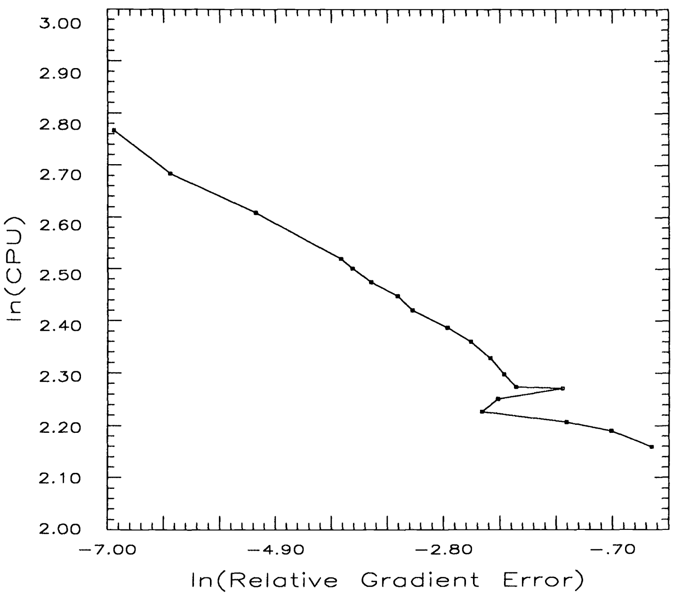


FIG. 12. *Solution time required for one gradient evaluation in tritium parameter identification problem.*

A somewhat different implementation of the trust region method was used rather than the Dennis–Schnabel code used in the last section. First, we included nonnegativity constraints on the first three components of x to be consistent with the physics of the problem. This was done by replacing the ellipsoidal trust region in (4) with the rectangular trust region $\|D_k s\|_\infty \le \Delta_k$, and by using a quadratic programming code at each iteration to exactly solve the trust region subproblem subject to the nonnegativity constraints on x. Second, we used the Hessian safeguarding techniques proposed in [2] in addition to the standard BFGS update (5).[5] Third, we used a simplified stopping criterion for comparison purposes. Each test case was terminated when $f_k - f^* \le \frac{1}{100}(f_0 - f^*)$, where f^* was the optimal value of f.

We remark that the convergence theory developed in [4] and [3] does *not* apply to algorithms that use nonellipsoidal trust regions or impose constraints on the variables; it is therefore interesting to investigate whether this modified implementation still performs well in practice in the presence of gradient errors.

Figure 13 shows the results of our tests.

Case 1 was tested for 15 different values of ζ_g. Note that the total computational time required is less than 8000 seconds for $\zeta_g = 0.015$, but rises to almost 16,000 and 24,000 seconds for $\zeta_g = 1.5 \times 10^{-5}$ and $\zeta_g = 0.25$, respectively. Fewer data were collected for cases 2 and 3, but note that the correction (45) appears to make

[5] This safeguarding procedure monitors the quantity $g_k^T B_k g_k / g_k^T g_k$ and compares it to the maximum observed problem curvature. If the quantity appears too large, B_k is "corrected" by performing a supplemental BFGS update involving one extra gradient evaluation at the point $x_k + s_\epsilon$, with $s_\epsilon = -\epsilon g_k$ for some small ϵ.

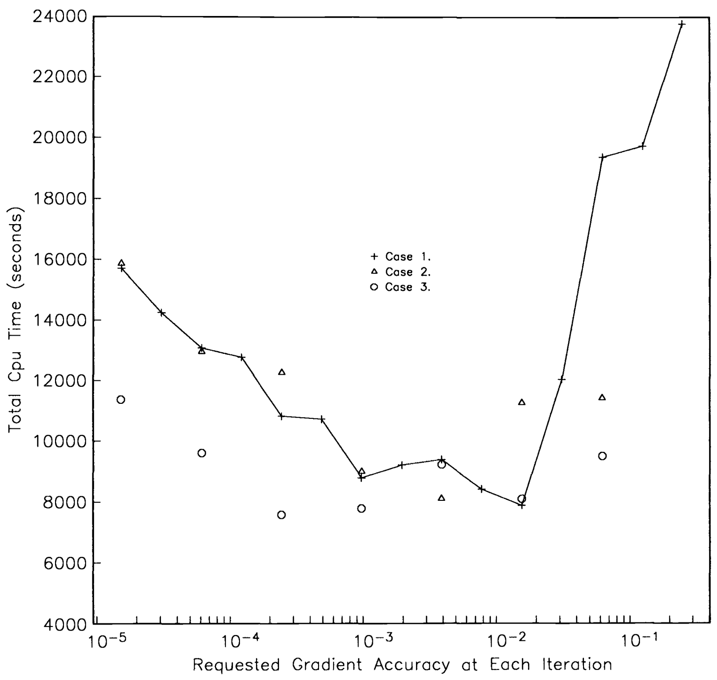


FIG. 13. *Solution time for numerical optimization of the tritium parameter identification problem.*

little difference to the algorithm when $e_k^T g_k / g_k^T g_k$ is small. On the other hand, using Procedure 2 rather than computing each f_k to a fixed accuracy of 10^{-8} resulted in a moderately faster algorithm.

In addition to the above three cases, a number of other numerical tests were made. Rather than keeping ζ_g fixed throughout the algorithm, we tried setting

$$\zeta_g^{k+1} = \begin{cases} 1/10 & \text{if } k = 0 \\ \max\{\zeta_g^k/2, 10^{-5}\} & \text{otherwise,} \end{cases} \tag{47}$$

or conversely,

$$\zeta_g^{k+1} = \begin{cases} 10^{-5} & \text{if } k = 0 \\ \min\{2\zeta_g^k\ , \ 1/10\} & \text{otherwise.} \end{cases} \tag{48}$$

The idea behind (47) is to try to obtain the fast local convergence properties of the BFGS method when accurate gradients are available, while the idea behind (48) is to avoid highly accurate gradient approximations near the solution where they are likely to be most expensive. Interestingly, both of these approaches performed similarly, requiring 8887 and 10,647 seconds, respectively.

Although the correction (45) appears to be of little use when $e_k^T g_k / g_k^T g_k$ is already small, it does appear to be useful in preventing the algorithm from failing due to occasionally encountering highly inaccurate gradient evaluations. In tests where an extremely large synthetic error ($\zeta_g \approx 1$) was added to the gradient approximation every p iterations, the algorithm was much more robust when (45) was used (although the algorithm still had problems involving convergence to a point with $g_k = 0$ and $\nabla f(x_k) \neq 0$, as predicted in [3]). In a similar vein, the gradient accuracy test given by (42) and (44) should be useful in verifying the accuracy of analytically derived gradients.

6. Summary. We have examined the numerical behavior of trust region algorithms for nonlinear optimization when function and gradient values are not computed exactly. This class of algorithms has proven remarkably robust, and can be successfully implemented even with very large errors in the function and gradient evaluations.

In a large number of tests using standard test problems with synthetically induced gradient errors, we observed that the algorithm performance, as measured by the number of iterations required for convergence, tends to degrade exponentially as the relative gradient error increases. This is a telling argument for using accurate evaluations provided they can be obtained at reasonable expense. For many optimization/simulation problems, however, the computational expense of these evaluations rises sharply with increasing accuracy, and low accuracy evaluations are again attractive. A good choice for the amount of relative gradient error allowed in the algorithm can result in orders-of-magnitude savings in computational cost. If (29) holds, then the choice $\zeta_g = l/4$ was nearly optimal for all of our test problems.

Using a parameter estimation problem based on the numerical solution of a system of ODEs, we tested a technique for estimating and controlling the amount of error in a gradient approximation. This technique was very successful when used in conjunction with the "user-specified accuracy" feature in the numerical differential equation solver ODEPACK. Actual computational costs for various values of relative gradient error were examined to confirm the behavior observed in the test problems with synthetically induced errors.

REFERENCES

[1] T. M. Apostol, *Mathematical Analysis*, Addison-Wesley, Reading, MA, 1957.

[2] R. G. Carter, *Safeguarding Hessian approximations in trust region algorithms*, Tech. Rep. TR87-06, Dept. of Mathematical Sciences, Rice University, Houston, TX, 1987.

[3] ———, *Numerical optimization in Hilbert space using inexact function and gradient information*, Tech. Rep. 89-45, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, 1989.

[4] ———, *On the global convergence of trust region algorithms using inexact gradient information*, SIAM J. Numer. Anal., 28 (1991), pp. 251–265.

[5] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[6] R. Fletcher, *Methods for nonlinear constraints*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, London, 1982, pp. 185–212.

[7] D. M. Gay, *Computing optimal locally constrained steps*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 186–197.

[8] A. C. Hindmarsh, ODEPACK, *a systematized collection of* ODE *solvers*, in Scientific Computing, R. S. Stepleman et al., eds., North–Holland, Amsterdam, 1983, pp. 55-64.

[9] J. D. Lambert, *Computational Methods in Ordinary Differential Equations*, John Wiley and Sons, New York, 1973.

[10] J. N. Lyness, *Remarks about performance profiles*, Tech. Memo. 369, Applied Mathematics Div., Argonne National Laboratory, Argonne, IL, 1981.

[11] V. A. Maroni, R. H. Land, and M. Minkoff, TSOAK-M1: *A computer code used to determine tritium reaction/adsorption/release parameters from experimental results of air-detritiation tests*, Rep. ANL-79-82, Argonne National Laboratory, Argonne, IL, 1979.

[12] M. Minkoff, *Approaches to optimization/simulation problems*, Appl. Numer. Math., 3 (1987), pp. 453–466.

[13] J. J. MORÉ, *Recent developments in algorithms and software for trust region methods*, in Mathematical Programming: State of the Art, A. Bachem, M. Grötschel, and B. Korte, eds., Springer-Verlag, Berlin, 1983, pp. 258–287.

[14] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Fortran subroutines for testing unconstrained optimization software*, Trans. Math. Software, 7 (1981), pp. 136–140.

[15] ———, *Testing unconstrained optimization software*, Trans. Math. Software, 7 (1981), pp. 17–41.

[16] M. J. D. POWELL, *A new algorithm for unconstrained optimization*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian, and K. Ritter, eds., Academic Press, London, 1970, pp. 31–65.

[17] G. A. SCHULTZ, R. B. SCHNABEL, AND R. H. BYRD, *A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties*, SIAM J. Numer. Anal., 22 (1985), pp. 47–67.

[18] O. TALAGRAND, P. C. SHAH, W. C. THACKER, I. M. NAVON, AND X. ZOE, *Part* V. *Adjoint equations for inverse problems*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991.

[19] PH. L. TOINT, *Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space*, IMA J. Numer. Anal., 8 (1988), pp. 231–252.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 389–405, March 1993

A MULTIRESOLUTION METHOD FOR DISTRIBUTED PARAMETER ESTIMATION*

JUN LIU[†]

Abstract. A multiresolution method for distributed parameter estimation (or inverse problems) is studied numerically. The identification of the coefficient of an elliptic equation in one dimension is considered as our model problem. First, multiscale bases are used to analyze the degree of ill-posedness of the inverse problem. Second, based on some numerical results, it is shown that the method of scale-by-scale multiresolution yields robust and fast convergence. Finally, it is shown how the method gives a natural regularization approach which is complementary to Tikhonov's regularization.

Key words. identification, distributed parameter, multiresolution, Haar basis, BFGS optimization

AMS(MOS) subject classifications. 35R30, 49D07, 65, 93B

1. Introduction.

Inverse problems are usually considered difficult problems. One essential question is how to obtain a stable solution rapidly. The idea of multiscale representation has suggested various parallel and iterative algorithms. For example, multigrid methods have been widely used for solving partial differential equations (direct problems) [4], [5] in order to accelerate the convergence. However, this idea has seldom been used until now for inverse problems (lack of theory). Recently, some approximation frameworks have been developed for inverse problems [2] and wavelet theory (multiscale bases) has appeared [14], [13]. This allows the idea of multiresolution to be used for inverse problems with little modification from the classical optimization methods.

We take as our model a simple elliptic problem that is well known as being nonlinear and ill posed. A number of studies of this problem have already been done (for example, [12]). In this paper, we first analyze the inverse problem using the Haar basis (a multiscale basis). We remark that the objective function is more "ill posed" (or nonlinear) with respect to the coefficients corresponding to the finer scales, but it is less sensitive with respect to these coefficients. Then we investigate numerically the behavior of a BFGS optimization routine in the IMSL library [11] when the unknown parameter is represented in the usual local basis, and in the Haar basis. This leads us to propose a multiresolution, or scale-by-scale, optimization method to solve the parameter estimation problem. This method turns out to be very robust (convergence is obtained for any initialization of the optimization algorithm) and efficient (a good solution is obtained with a very small number of iterations). The multiresolution method allows us to perform, for the problem under consideration, a global optimization with a local optimizer. Moreover, a method of rescaling of the variables and the objective function for each optimization run can be easily performed to accelerate the convergence. Finally we give regularization methods that are complementary to Tikhonov's regularization. This paper emphasizes some numerical results and does not give a global convergence proof.

*Received by the editors February 13, 1991; accepted for publication (in revised form) December 12, 1991.

[†]Ident Project, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, 78153 Le Chesnay, France. Present address, Fluides, Automatiques et Systèmes Thermique (F.A.S.T.), URA871 au Centre National de la Recherche Scientifique, Batiment 502, Campus Universitaire, 91405 Orsay, France.

2. A simple model inverse problem.

We consider the following elliptic equation:

$$-\frac{d}{dx}\left(a(x)\frac{du(x)}{dx}\right) = f(x), \qquad x \in (0,1) \tag{1}$$

with the Dirichlet boundary condition

$$u(0) = u(1) = 0, \tag{2}$$

where $f(x) \in L^2(0,1)$ is known.

The direct problem is:

(DP) let Q_{ad} be the convex set $\{a(x)\,|\,0 \le m \le a(x) \le M\ , x \in (0,1)\} \subset L^2(0,1)$; given $a(x) \in Q_{ad}$, we solve for $u(x)$ in (1), (2), where *ad* denotes admissible.

This problem is very simple, as (1), (2) have a unique solution $u(x)$ denoted by $\Phi(a)$. The application Φ is continuous from Q_{ad} equipped with the L^2-norm into $H_0^1(0,1) \subset L^2(0,1)$.

The inverse problem is:

(IP) given a distributed observation $z(x) \in L^2(0,1)$ of $u(x)$, we minimize the objective function (output least squares error)

$$J(a) = \|\Phi(a) - z\|^2 = \int_0^1 (u(x) - z(x))^2 dx \tag{3}$$

over $a \in Q_{ad}$.

The problem is a typical ill-posed nonlinear problem. There are two types of "ill-posedness." First, $a(x)$ cannot be determined by (1) on the set $\{x|u_x(x) = 0\}$ so that there is no uniqueness for the solution if $measure(\{x|u_x(x) = 0\}) > 0$. Second, from homogenization theory [3], we remark that there is a sequence $a_n(x)$ which is not convergent in L^2, but still the sequence $u_n(x) = \Phi(a_n(x))$ is convergent in $H_0^1(0,1)$, so that the solution is not stable.

Remark. We can easily obtain an analytical solution of (1), (2):

$$u(x) = \frac{\int_0^1 b(y)F(y)dy \int_0^x b(y)dy}{\int_0^1 b(y)dy} - \int_0^x b(y)F(y)dy \tag{4}$$

with

$$b(x) = a(x)^{-1}, \qquad F(y) = \int_0^y f(s)ds. \tag{5}$$

This analytical solution will be used in §3.2.

For the numerical solution of (IP), we shall consider the following discretized problem (IP_h^H):

$$\text{Minimize} \quad J_h(a_H) = \sum (u_h - z_h)^2 \tag{6}$$

with

$$A_h(a_H)u_h = f_h, \tag{7}$$

where u_h and a_H are the discretizations of $u(x)$ and $a(x)$, respectively ($h = 1/n$ and $H = 1/N$); u_h is a piecewise linear function which consists of the $n-1$ values of u_h

at the nodes $x_i = ih$, $i = 1, \dots, n-1$; and a_H is a piecewise constant function which consists of the N values of a_H on intervals $](i-1)H, iH[$, $i = 1, 2, \dots, N$.

Remark. For well-posedness of the discrete problem (IP_h^H), the condition $h \le H$ is necessary. When $H = h$, we have $N = 1/H$ numbers to be estimated, but only $N-1$ observation numbers.

3. Using multiscale bases to analyze the inverse problem.

3.1. Multiscale bases. Multiscale bases have been known for a very long time. The first one that appeared in the literature was the Haar basis, presented at the beginning of this century. They were, however, relatively little used for numerical computations, until the general interest in wavelets developed in the early 1980s. Presented first as a challenge to Fourier analysis, wavelets quickly turned out to be a systematic way of constructing multiscale bases of function spaces. The use of these multiscale bases is now being widely investigated for the resolution of partial differential equations (direct problem). We show in this paper that they can be a very valuable (and simple) tool for the resolution of the inverse problem stated in §2. Because the parameter $a(x)$ that we are looking for is required only to be in $L^2(0,1)$, we shall use the simplest multiscale basis, namely the Haar basis. Of course, in inverse problems where the unknown parameter is more regular, we must resort to more sophisticated wavelet bases.

Let Z be the set of integers, using the characteristic function

$$\phi(x) = \begin{cases} 1, & x \in (0,1), \\ 0, & \text{otherwise.} \end{cases}$$

The mth scale approximation of a general function $a(x)$ is represented by

$$a^m(x) = \sum_{i \in Z} a_i^m \phi(2^m x - i), \tag{8}$$

where a_i^m is the mean value of $a(x)$ over the interval $[i/2^m, (i+1)/2^m]$.

The corresponding multiscale basis is the Haar basis, made of the functions $\psi_i^j(x) = \psi(2^j x - i)$ for all $i, j \in Z$, which are constructed from the following mother wavelet function:

$$\psi(x) = \begin{cases} -1, & x \in (0, \frac{1}{2}), \\ +1, & x \in (\frac{1}{2}, 1), \\ 0, & \text{otherwise.} \end{cases}$$

The ψ_i^j, $i, j \in Z$ form a complete orthogonal basis of $L^2(R)$, which is multiscale in the sense that the mth scale approximation $a^m(x)$ is simply obtained by setting to zero all coefficients of ψ_i^j with $j \ge m$ in the expansion of the function

$$a(x) = \sum_{i \in Z, j \in Z} c_i^j \psi_i^j(x) \tag{9}$$

on the basis.

Remark. For the space $L^2(0,1)$ and a fixed mth scale approximation, we have two equivalent orthogonal bases: the characteristic basis made of $\{\phi(2^m x - i + 1) \mid i = 1, \dots, 2^m\}$ and the Haar basis made of $\{\psi(2^j x - i + 1) \mid j = 0, \dots, m-1, i = 1, \dots, 2^j\}$ and of the constant function $\psi^0(x) = \phi(x) = 1$ over [0,1].

3.2. Curvature analysis. A geometrical theory for general nonlinear least-squares problems [7] shows that the velocity and the curvature along curves of the solution space which are images by Φ of segments in Q_{ad} are important to measure the "degree of ill-posedness" of the inverse problem in the output least-squares formulation. In the limiting case, the curvature is equal to zero for a linear problem.

In this section, we suppose that $f(x) = 1$. Algebraic computation systems (Macsyma, Maple, etc.) allow us to calculate the velocity and the curvature along the curve $\Phi(a + t\delta a)$ at $t = 0$ in the directions of the basis functions

$$\delta a(x) = \begin{cases} 0, & x \in (0, t-h), \\ -1, & x \in (t-h, t), \\ 1, & x \in (t, t+h), \\ 0, & x \in (t+h, 1), \end{cases}$$

for the exact mapping Φ, i.e., without any approximation to the solution $u(x)$ of (1) and (2).

For the velocity V and the curvature K in the direction $\delta a(x)$ at $a(x) = 1$, we obtain the following formulae:

$$V^2 = \frac{h^3(180hs^2 + 5h + 30h^3 + 40s^2 - 24h^2)}{60}, \tag{10}$$

$$\begin{aligned} K^2 = -480(&1230h^7 - 600h^8 - 940h^6 + 317h^5 - 40h^4 + 1800h^7s^2 - 400s^4 \\ &-3000h^6s^2 + 7200h^4s^4 + 2640h^4s^2 + 480h^2s^4 + 240h^2s^2 \\ &-1480h^3s^2 - 8400h^3s^4 - 330h^5s^2 + 7200hs^6 + 1600hs^4 - 4800s^6) \\ \times(&180hs^2 + 5h + 30h^3 + 40s^2 - 24h^2)^{-3}h^{-4}, \end{aligned} \tag{11}$$

where $s = t - \frac{1}{2}$.

From these formulae, we have

$$V^2 \sim \frac{2h^3s^2}{3}, \quad K \sim 6h^{-2}, \quad \text{as} \quad h \to 0. \tag{12}$$

We conclude that the objective function is more nonlinear with respect to the coefficients of the finer scales, but is less sensitive with respect to these coefficients. For illustration, we plot the curves $\log(V)$ and $\log(1 + K^2)$ at five different scales in Fig. 1 (bottom).

Remark. In the same way, for the characteristic basis we have

$$V^2 = \frac{h^2(h^3 + 30h^2s^2 - 20hs^2 + 10s^2 + 120s^4)}{120}, \tag{13}$$

$$K^2 = -\frac{4800h^3s^2(10hs^2 + 2h - 12s^2 - 1 - h^2)}{(h^3 + 30h^2s^2 - 20hs^2 + 10s^2 + 120s^4)^3}, \tag{14}$$

$$V^2 \sim \frac{h^2s^2(1 + 12s^2)}{12}, \quad K^2 \sim \frac{4.8h^3}{s^4(1 + 12s^2)^2} \quad \text{as} \quad h \to 0 \tag{15}$$

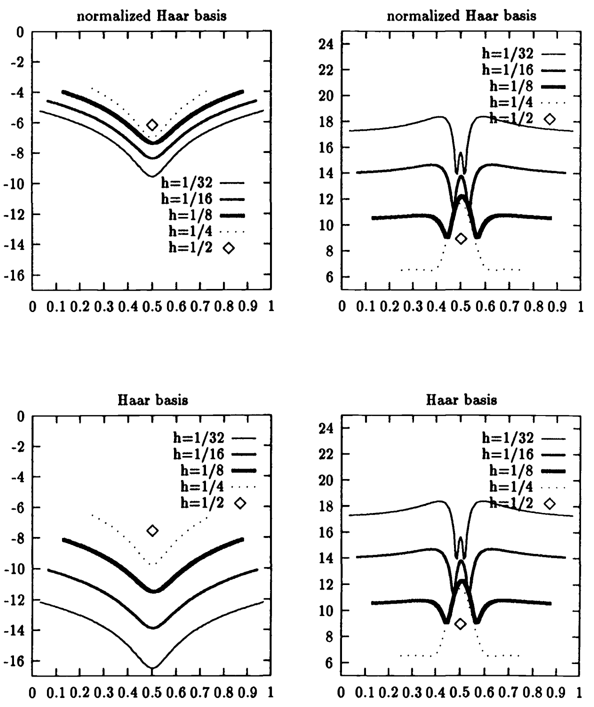


FIG. 1. *Left:* $\log(V^2)$ *and right:* $\log(1+K^2)$ *at different scales.*

in the direction of the basis function

$$\delta a(x) = \begin{cases} 0, & x \in (0, t-h/2), \\ 1, & x \in (t-h/2, t+h/2), \\ 0, & x \in (t+h/2, 1), \end{cases}$$

where $s = t - \frac{1}{2}$.

Remark. If we take $b(x) = a(x)^{-1}$ as the parameter to be estimated, we do not get the same result. The curvature K in the direction $\delta b(x)$ at $b(x) = 1$ is the following:

$$K^2 = \frac{9600h^2(5-24h+60s^2+30h^2-90hs^2)}{(5h+40s^2-24h^2+180hs^2+30h^3)^3}, \tag{16}$$

$$K^2 \sim \frac{3h^2(1+12s^2)}{4s^6} \quad \text{as} \quad h \to 0, \tag{17}$$

where $s = t - \frac{1}{2}$ and

$$\delta b(x) = \begin{cases} 0, & x \in (0, t-h), \\ -1, & x \in (t-h, t), \\ 1, & x \in (t, t+h), \\ 0, & x \in (t+h, 1). \end{cases}$$

It is not surprising that the values of K in (12), (15), and (17) are very different. Equation (12) shows that singularity directions for $K \to \infty$ correspond to $h \to 0$ and from (15) and (17), singularity directions for $K \to \infty$ correspond to $h \to 0$ and $s \sim h$. These two kinds of singularity directions correspond exactly to the two kinds of ill-posedness mentioned in §2, which can also be recovered by the singularity directions for $V \to 0$ from (12). So the inverse problem is not only insensitive but also very nonlinear for the singularity directions.

4. Multiscale parametrization, multiresolution, and regularization. In this section we investigate various strategies for the practical computation of a_H by minimization of the objective function J_h. In all runs we have used the subroutine BCONF of the IMSL library based on the BFGS algorithm [11]. The only initializations required in this subroutine are the initial guess of the unknown parameter. The other parameters are given by default values (the initialization of the Hessian is an identity matrix).

Our investigations concerned first the behavior of the BFGS algorithm for various initializations and various parametrizations of a_H associated to different bases of R^{32}: the characteristic (or local) basis (§4.1) and multiscale bases (§4.2). In many cases, parametrization in the usual Haar basis will prove to yield much better convergence of the BFGS algorithm than the usual local basis.

Our analysis of this phenomenon will lead us to experiment with a multiresolution (scale-by-scale) optimization approach in §4.3, which turns out to be very robust and yields the fastest convergence. Finally, we show in §4.4 how multiscale parametrization and multiresolution can be efficiently combined with a regularization approach in order to stabilize the estimation of a_H with noisy data.

Here we summarize the data common to all runs: the elliptic equation (1), (2) (with right-hand side $f(x) = 1$) was discretized using a finite difference scheme with a mesh size $h = 1/32$, and the parameter a_H was also discretized with $H = 1/32$ (hence we are faced with the estimation of 32 unknown parameters).

The data z_h (exact in §4.1, 4.2, and 4.3 and noisy in §4.4) were generated using the above finite difference scheme and the following "true parameter"

$$a_H^{\text{true}}(x) = \begin{cases} 1, & x \in (0, 1/5), \\ 10, & x \in (1/5, 2/3), \\ 50, & x \in (2/3, 1). \end{cases}$$

Note that a_H^{true} is quite strongly heterogeneous! Because of the Dirichlet boundary condition, z_h consisted of only 31 numbers representing the solution $u(x)$ at $x = h/2, 3h/2, \ldots, 63h/2$. Hence the problem of the estimation of a_H from z_h was clearly undetermined!

Two initializations were used, corresponding to $a_H^{\text{init}} \equiv 1$ and $a_H^{\text{init}} \equiv 10$. The first initial guess is "poor" (underestimated), and the second one is "good" (in the range of the values of a_H^{true}). The corresponding initial values of J_h are 215 and 0.84, respectively. Optimization runs using the local basis were made with the bound constraint of $0.1 \le a_H \le 100$ (except one run in §4.2), and optimization runs using the multiscale basis or using the multiresolution method were performed without any constraint.

4.1. The BFGS algorithm and the local basis. The traditional method is to perform one optimization run to simultaneously estimate 32 numbers representing the value of a_H on each interval. If we use the subroutine BCONF in this way, we obtain the result shown at the bottom left of Fig. 2 after 1000 iterations with

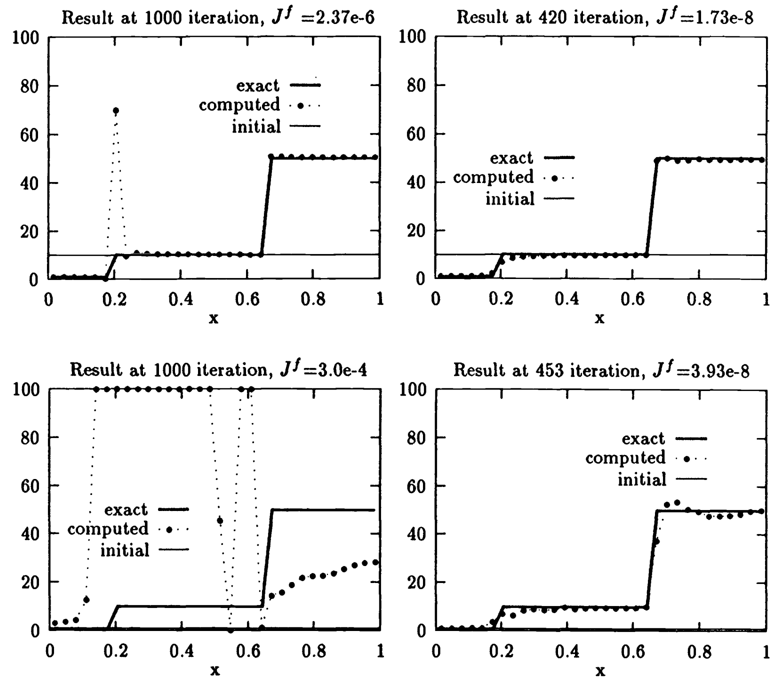


FIG. 2. *The comparison of the final results of minimization for different initializations (bottom $a_{\text{init}} = 1$, top $a_{\text{init}} = 10$) and for different parametrizations of the unknown parameter (left: local basis, right: Haar basis). The bound constraint* $[0.1, 100]$ *was used with the local basis and no constraint was used with Haar basis.*

initialization $a_H^{\text{init}} = 1$. The algorithm is not convergent and the final relative objective function (i.e., the ratio of the final value of J_h to the initial value of J_h: $J^f = J_h(a_H^{\text{comp}})/J_h(a_H^{\text{init}})$) is equal to 3.0×10^{-4}. If we choose the better initialization $a_H^{\text{init}} = 10$, we obtain the result shown at the top left of Fig. 2. We have recovered the "true" parameter a_H^{true}, but the rate of convergence is slow. To understand in more detail the behavior of the algorithm on the problem, we plot the evolution of the relative errors for parameters at different scales in L^2 during the procedure of optimization in Figs. 3 and 4. The result coincides with the analysis in §3. The coefficients of finer scales do not converge toward the "true" coefficients at all when the "bad" initialization $a_H^{\text{init}} = 1$ is used.

4.2. The BFGS algorithm and multiscale bases. We tested the influence of the representation of a_H on the Haar basis (orthogonal) as in (9), on the normalized Haar basis (orthonormal), and on the local basis (orthonormal) when a minimization without constraints was performed for the three bases. Note that quasi-Newton algorithms are expected to produce the same sequence of iterates when an orthonormal change of basis is performed on the unknown parameter and when the Hessian is initialized with an identity matrix in both cases. Hence one expects that using the local basis or the normalized Haar basis will not influence the behavior of the BFGS algorithm. This is what we observe in Fig. 5. However, the decrease of the objective function is slightly better with the normalized Haar basis for the two initializations $a_H^{\text{init}} = 1$ and $a_H^{\text{init}} = 10$.

Then we used the usual Haar basis for the representation of the unknown parameter a_H. As we already noted in [8], the convergence of the BFGS algorithm was

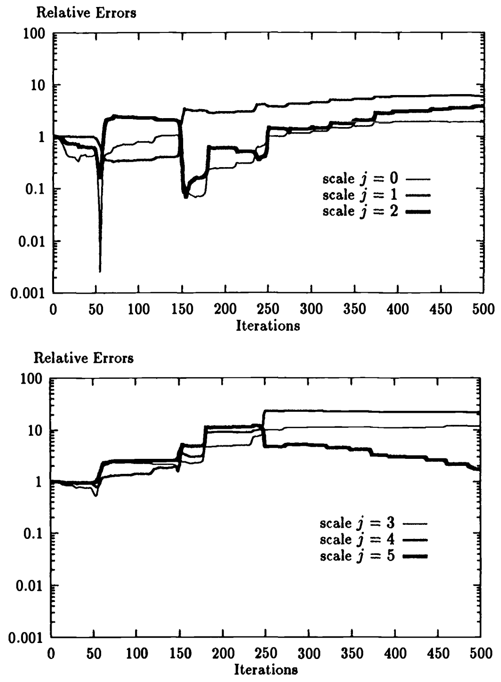


FIG. 3. *The evolution of relative errors on different scales for parameter $a(x)$ in L^2 during the optimization with the local basis, $a_{\text{init}} = 1$ and bound constraint $= [0.1, 100]$.*

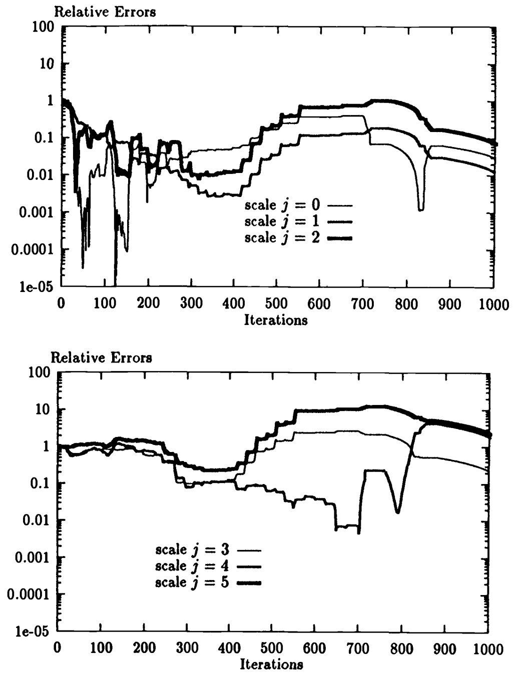
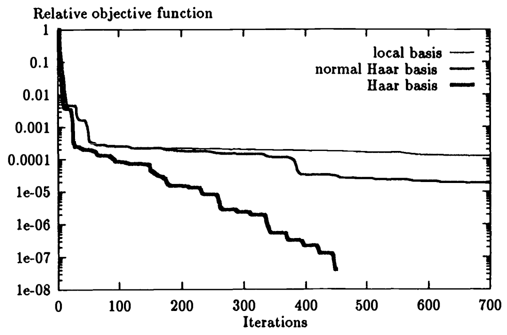


FIG. 4. *The evolution of relative errors on different scales for the parameter $a(x)$ in L^2 during the optimization with the local basis, $a_{\text{init}} = 10$ and bound constraint $= [0.1, 100]$.*



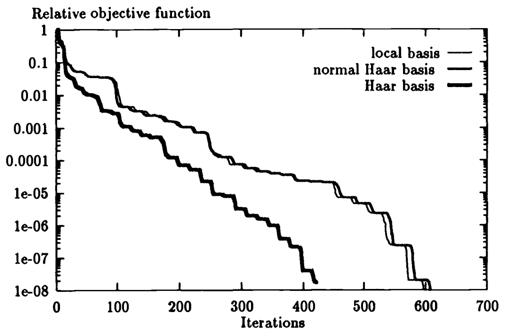
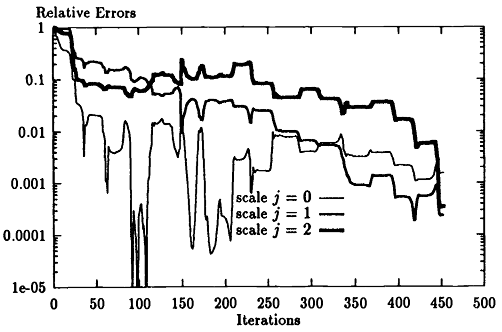


FIG. 5. *Convergence of the objective function obtained with local, normal Haar and Haar bases for* $a_{\text{init}} = 1$ *(top) and* $a_{\text{init}} = 10$ *(bottom) (no constraint was used).*



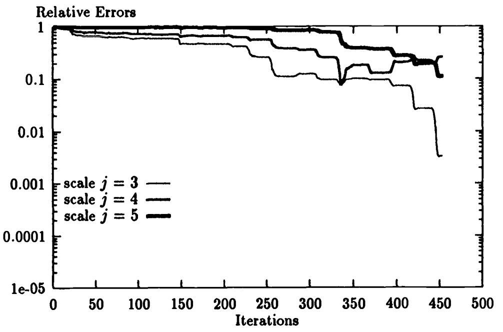
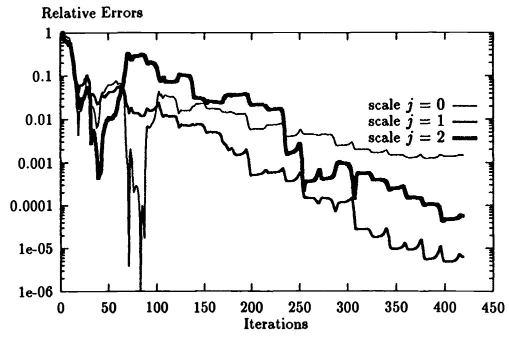


FIG. 6. *The evolution of relative errors on different scales for parameter* $a(x)$ *in* L^2 *during the optimization with the Haar basis and* $a_{\text{init}} = 1$.



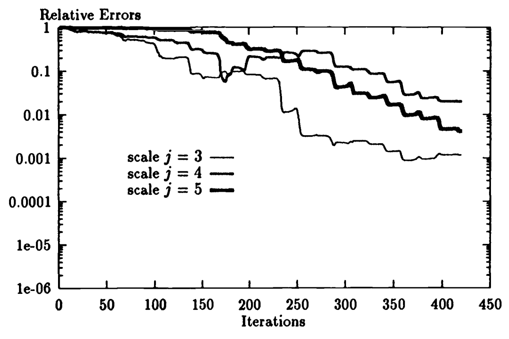


FIG. 7. *The evolution of relative errors on different scales for parameter $a(x)$ in L^2 during the optimization using the Haar basis with $a_{\text{init}} = 10$.*

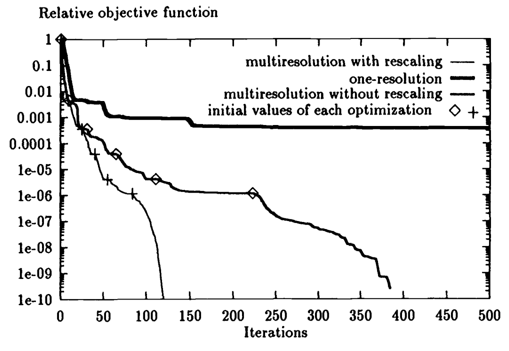


FIG. 8. *Convergence of the objective function obtained by one-resolution, multiresolution with rescaling and without rescaling for each optimization run (the local basis and $a_{\text{init}} = 1$ were used in the three cases).*

much better with this basis for the two initializations $a_H^{\text{init}} = 1$ and $a_H^{\text{init}} = 10$ (Fig. 5). We now try to give an explanation for this phenomenon. We note first that in the Haar basis, the L^2-norm of the basis function is decreasing when the scale index is increasing. Hence replacing the local or normalized Haar basis by the Haar basis amounts to performing a rescaling of the unknown parameter depending on the scale level: the sensitivity to an unknown is decreased proportionally to its scale index (this is apparent by comparing the top and bottom of the left part of Fig. 1). Surprisingly at first glance, this rescaling goes in the opposite direction of what would be required to try to "spherize" the objective function J_h around its minimum: the discrepancy in sensitivity in all basis directions is much larger with the Haar basis (bottom left of Fig. 1) than with the normalized Haar basis (top left of Fig. 1)! Here "spherizing" means the scaling of the diag($\nabla^2 J_h(a)$). Numerical experimentation with an "overnormalized" Haar basis (for which all scales had approximately the same sensitivity) did not produce any enhancement of the convergence.

The explanation may come from the nonlinear effects. As we already mentioned in §3.2, the nonlinearity is increasing with scale index, hence "spherizing" the objective function, which boosts up the sensitivity of fine (high-index) scales, and also boosts up the nonlinear effects associated with these fine scales, so that the "convergence domain" around the exact solution becomes extremely small, and the quasi-Newton method is not able to find its way to the global minimum once the initial guess is not too good. Conversely, "despherizing" the objective function, i.e., diminishing the sensitivity of fine (high-index) scales (which is performed by using the Haar basis), also lowers the influence of the strong nonlinearities associated with fine-scale coefficients: at the beginning, the objective function seems to depend almost entirely on the low scales having low nonlinear coefficients. It is only after these low-scale coefficients have been approximately set that the influence of the finer-scale coefficients becomes apparent. This is confirmed in Figs. 6 and 7, where the evolution of the relative errors on a_H is plotted for each scale against the iteration index. On Fig. 6 for example, we see that coefficients associated to scales 1 and 2 begin to adjust only after iteration 20, when the scale zero (mean value) coefficient has been approximated to 10 percent. Similarly, until iteration 150, the BFGS algorithm, does not touch the coefficient of the finer scales (third, fourth, and fifth), and hence performs an optimization in the four-dimensional space associated with scales 0, 1, and 2! Furthermore, we see that the coefficients of the finest (fifth) scale really begin to adjust only after iteration 340, where the error on all coarser scales becomes less than 10 percent!

This behavior strongly suggests trying to optimize the objective function successively on spaces of increasing dimension associated with finer and finer scales. This is the subject of the next section.

4.3. Multiresolution algorithms. As we have seen above, the nonlinear effects are increasing when finer scales are added, and the size of the corresponding "domain of convergence" of the quasi-Newton method is decreasing. It is hence natural to solve first the optimization problem on a small number of (coarser) scales, which is likely to yield the global minimum as the nonlinear effects are small, and to use this point as initial guess for an optimization run including finer scales. The hope is that the initial guess from the coarser optimization will be inside the "convergence domain" of the new, more nonlinear problem. This is the basis of the multiresolution algorithm.

Another argument comes from estimates of approximation error. In the solution of the partial differential equation (direct problem), we have the following error estimate [1], [9]:

$$\|u_h - u\| \leq K h^{\alpha}. \tag{18}$$

This error estimate is usually considered as essential for the full multigrid methods (nested algorithms) [5]. For inverse problems, an approximation theory has also been developed recently for elliptic problems in some special cases [2], [10] under the condition $h \sim H$, which gives (for nonnoisy data) a similar estimate:

$$\|a_H - a\| \leq K_2 H^{\beta}. \tag{19}$$

This also suggests using a nested algorithm for our inverse problem.

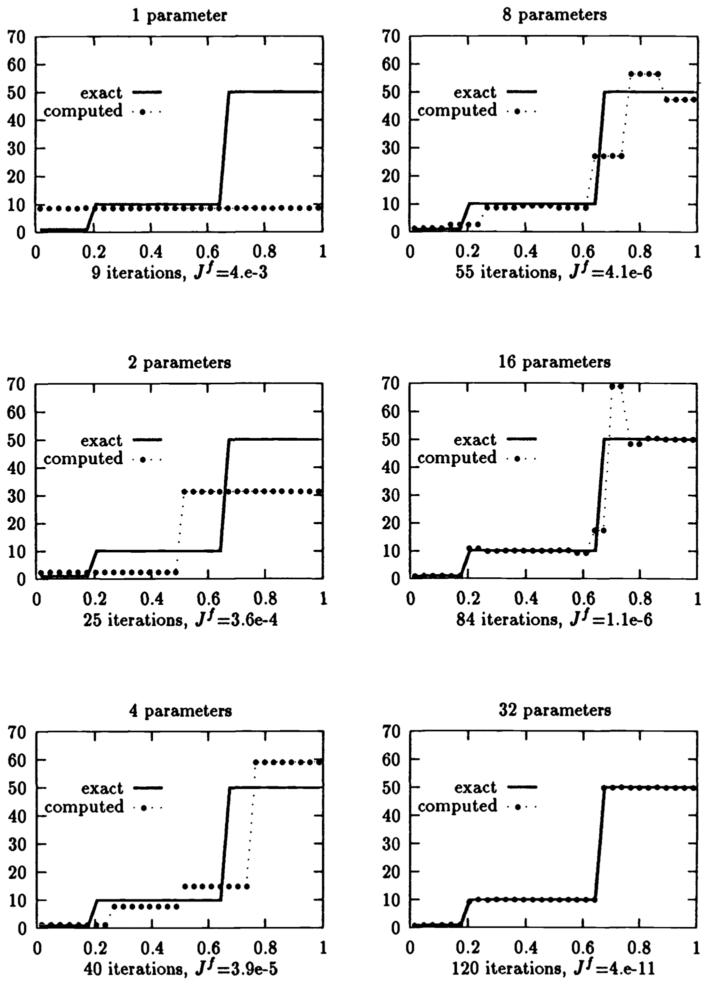


FIG. 9. *The parameter obtained at the end of each step (scale level) of the multiresolution method without noise.*

The multiresolution algorithm is as follows. We solve first the optimization problem with scale zero (one unknown), and the result is used as initial value for an optimization run with scale one (two unknowns), etc. For each optimization run, we use a rescaling method to get better convergence. The rescaling method is to reformulate the optimization problem Min$J(a)$ with initial value a^0 (suppose $a_i^0 \neq 0$, for all i) as Min$\hat{J}(b) = J(b{*}a^0)/J(a^0)$ with the initial value $b^0 = [1, 1, \ldots, 1]^T$, where $b{*}a^0 = [b_1 a_1^0, b_2 a_2^0, \ldots]^T$. The Hessian matrix is initialized as an identity matrix.

This algorithm can be implemented using either the local basis or the Haar basis. The advantage of the local basis is that it allows an easy implementation of the bound constraint, but our numerical experiments show that with the multiresolution algorithm, the solution does not tend to hit the constraints. The multiscale basis allows us to easily perform multigrid patterns such as V, W cycles [5]. But the inverse problem does not have the same behavior as the direct problem and until now we have not found any need to use such patterns.

We illustrate in Figs. 8, 9, and 10 the behavior of the local basis implementation of the multiresolution algorithm. Figure 8 shows that multiresolution achieves an

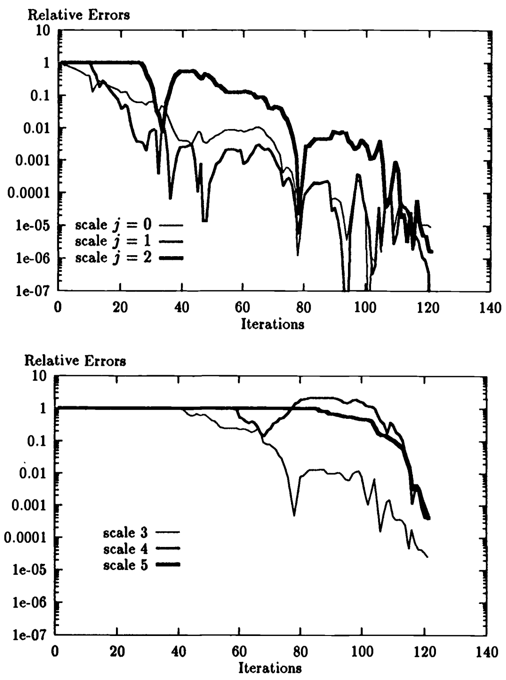


FIG. 10. *The evolution of relative errors on different scales for parameter $a(x)$ in L^2 during the optimization with $a_{\text{init}} = 1$ and bound constraint $[0.1 : 100]$ using multiresolution.*

excellent fit (the final objective function $= 10^{-10}$) in only 120 iterations, and clearly beats (mono)resolution with either the local or the Haar basis (compare, also, with Fig. 5). The rescaling for each optimization has the effect of reducing the number of iterations from 380 to 120 (by a factor of 3!) on the example considered. Figure 9 shows the parameter obtained at the end of each of the six optimization runs (scales 0, 1, 2, 3, 4, and 5), and Fig. 10 shows the evolution of the relative error in the parameter associated with each scale.

4.4. Multiscale bases, multiresolution, and regularization. When the data are noisy, the error estimate becomes [1], [9]

$$\|a_H - a\| \le K_1 \mathrm{dist}(z, \phi(Q_{ad}))H^{-s} + K_2 H^{\beta}. \tag{20}$$

So a_H may be far from a when H is sufficiently small. To stabilize the solution of an inverse problem, the classical method is to use Tikhonov's regularization [15], for example, replacing the objective function $J(a)$ by

$$J_1(a) = J(a) + \alpha \left\| \frac{da(x)}{dx} \right\|^2. \tag{21}$$

This function, of course, can be efficiently minimized using the multiresolution algorithm of §4.3. The numerical results corresponding to $\alpha = 10^{-6} J_h(a_H^{\text{init}})$ and 5 percent noisy data are shown in Fig. 11. We see that a stabilization is achieved at the price of a less accurate fit to the true parameter values.

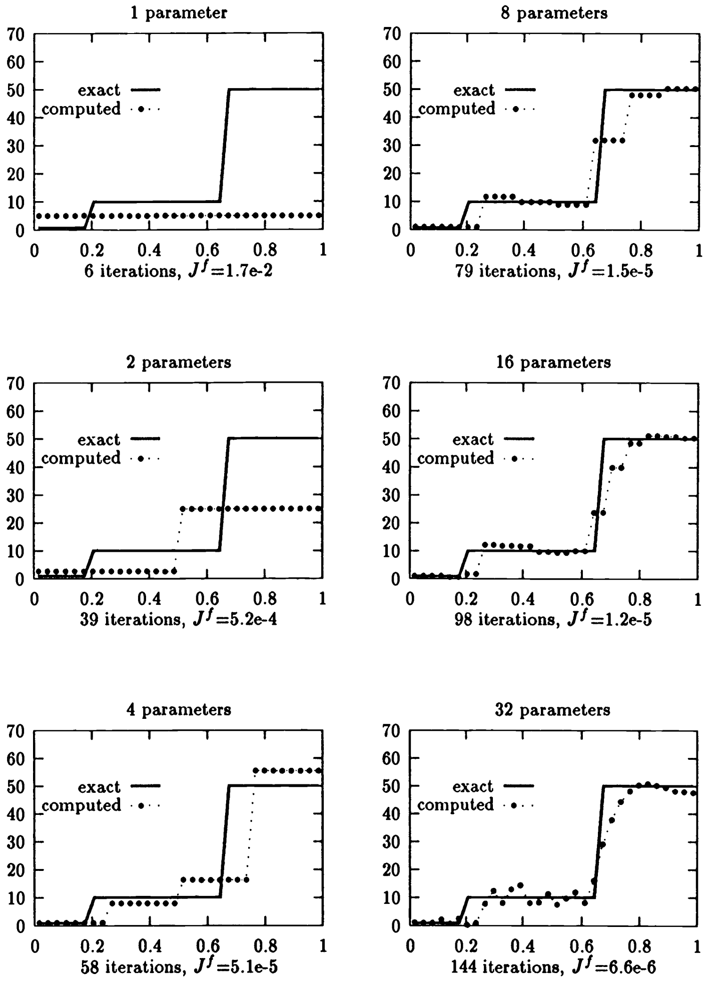


FIG. 11. *The parameter obtained at the end of each step (scale level) of the multiresolution method applied to Tikhonov's regularized problem and data with* 5 *percent noise.*

The use of the multiscale basis and/or the multiresolution algorithm leads naturally to two other types of regularization.

If we use a multiscale basis, we can express the smoothness of a function by the decay of the coefficient in the expansion. So we can replace $J(a)$ by (cf. [8])

$$J_2(a) = J(a) + \sum_j \alpha_j \sum_i (c_i^j)^2. \tag{22}$$

Hence the regularization is obtained by limiting the amplitude of the small-scale oscillations of a_H (compare with the usual regularization (21), where all scales are affected!). We refer to [8] for numerical results for this approach.

With the procedure of multiresolution, a very simple type of regularization (corresponding to the limit case of (22) with $\alpha_j = 0, j = 0, \dots, m$, and $\alpha_j = \infty$, $j > m$) can be performed simply by stopping at a reasonable scale m during the multiresolution procedure. Different regularizations can be obtained using different multiscale bases. For example, it is interesting to use some regular multiscale basis as in Meyer [14].

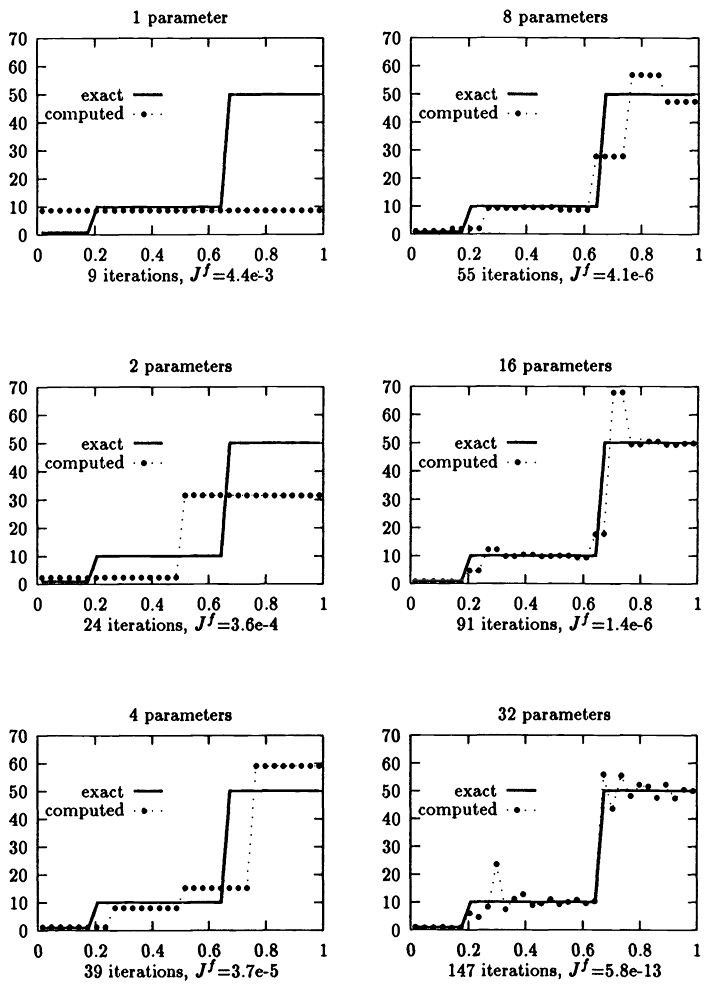


FIG. 12. *The parameter obtained at the end of each step (scale level) of the multiresolution method applied to the unregularized problem and data with* 1 *percent noise.*

Figures 12 and 13 show the parameters obtained at different scales for 1 percent noise and for 5 percent noise, respectively. As the noise increases, the oscillation increases also, hence we should stop the optimization procedure at an adequate scale to regularize the solution. For example, we can stop at the fourth scale (16 numbers) for 1 percent noise and at the third scale (eight numbers) for 5 percent. We remark that the solutions up to the third scale are the same with 5 percent noise or without noise (Figs. 9, 12, and 13).

5. Conclusion. We have numerically studied the use of a multiresolution approach for the inverse problem in a one-dimensional elliptic equation. It has been shown that the methodology of multiresolution is well suited to solving this ill-posed nonlinear problem. The advantages of this method are:

- The final solution is independent of the choice of the initial parameters;
- Some local minima can be avoided;

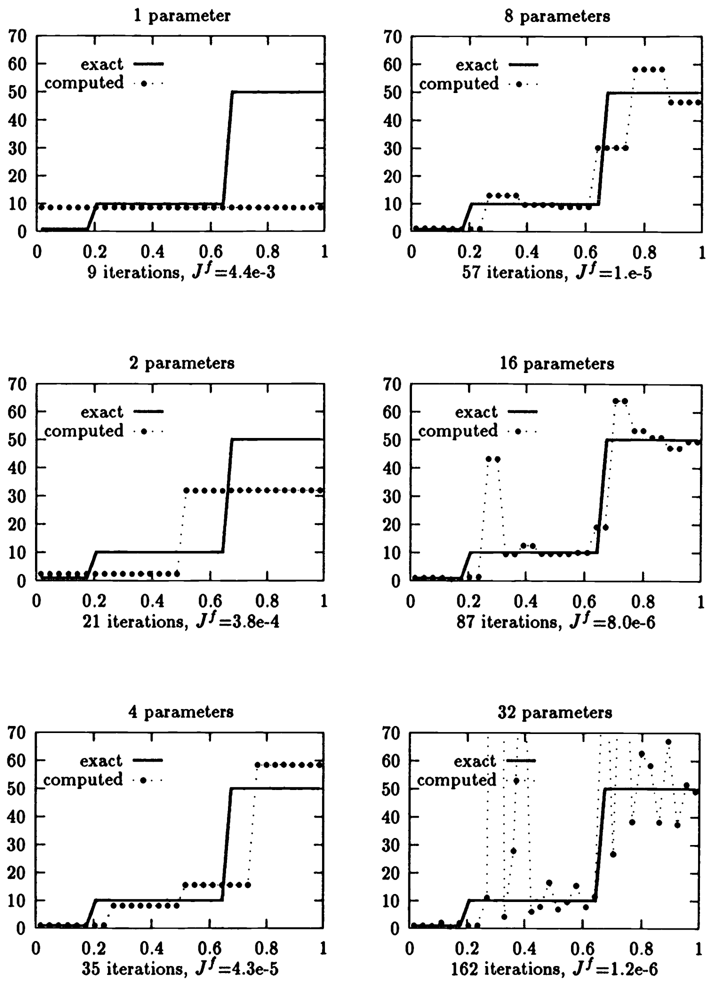


FIG. 13. *The parameter obtained at the end of each step (scale level) of the multiresolution method applied to the unregularized problem and data with* 5 *percent noise.*

- Very fast convergence is achieved for the numerical examples considered; and
- It leads naturally to the use of new types of regularizations which are likely to perturb the coarse-scale components of the optimal solution less.

The methodology proposed in this paper is very general. It has also been used for the identification of the conductivity coefficient in a parabolic equation [8], and for the identification of the relative permeabilities and capillary pressure [6]. However, many theoretical problems are still open. We believe that this method will also prove to be powerful for more complicated inverse problems.

Acknowledgments. I am greatly indebted to Professor Guy Chavent for many helpful conversations, and I would also like to thank the referee for many useful comments.

REFERENCES

[1] J. P. AUBIN, *Approximation of Elliptic Boundary-Value Problems*, John Wiley, New York, 1972.

[2] H. T. BANKS AND K. KUNISCH, *Estimation Techniques for Distributed Parameter Systems*, Birkhäuser, Boston, Basel, Berlin, 1989.

[3] A. BENSOUSSAN, J. L. LIONS, AND G. PAPANICOLAOU, *Asymptotic Analysis for Periodic Structures*, North-Holland, Amsterdam, 1978.

[4] A. BRANDT, *Multi-level adaptive solutions to boundary value problems*, Math. Comp., 13 (1977), pp. 333–390.

[5] W. BRIGGS, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[6] C. CHARDAIRE, G. CHAVENT, J. JAFFRÉ, AND J. LIU, *Multiscale representation for the simultaneous estimation of relative permeabilities and capillary pressure*, in SPE Fall meeting, New Orleans, LA, Sept., 1990.

[7] G. CHAVENT, *A nonlinear least-square theory for inverse problems*, in Inverse Methods in Action, Springer-Verlag, Berlin, 1990.

[8] G. CHAVENT AND J. LIU, *Multiscale parametrization for the estimation of a diffusion coefficient in elliptic and parabolic problems*, in Fifth IFAC Symposium on Control of Distributed Parameter Systems, Perpignan, France, June 1989.

[9] P. G. CIARLET, *The Finite Element Method of Elliptic Problems*, North–Holland, Amsterdam, 1978.

[10] R. S. FALK, *Approximation of Inverse Problems*, in Inverse Problems in Partial Differential Equations, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 5–14.

[11] IMSL INC., IMSL *Library Reference Manual, Version* 10, Houston, TX, 1987.

[12] K. KUNISCH AND L. W. WHITE, *Identifiability under approximation for an elliptic boundary value problem*, SIAM J. Control Optim., 25 (1987), pp. 279–297.

[13] S. G. MALLAT, *Multiresolution approximation and wavelet orthonormal bases of L^2*, Trans. Amer. Math. Soc., 1989, pp. 69–87.

[14] Y. MEYER, *Ondelettes et Opérateurs*, Hermann, Paris, 1990.

[15] A. N. TIKHONOV AND V. Y. ARSENIN, *Solution of Ill-Posed Problems*, John Wiley, New York, 1977.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 406–423, March 1993

A PARALLEL IMPLEMENTATION OF AN ITERATIVE SUBSTRUCTURING ALGORITHM FOR PROBLEMS IN THREE DIMENSIONS*

BARRY F. SMITH[†]

Abstract. Numerical results from a parallel implementation of a class of iterative substructuring algorithms are reported. The algorithms are for solving self-adjoint elliptic partial differential equations in three dimensions. Results are given for several variants of the algorithm. In the first variant, exact interior solvers are used; in the second, one multigrid V-cycle is used to solve the interior problems approximately. The results are compared with theoretical behavior of the algorithm reported in previous work. A numerical experiment involving the equations of linear elasticity is also included.

Key words. domain decomposition, finite elements, iterative substructuring, parallel computing

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

Much of the work on domain decomposition algorithms has focused on the abstract analysis of the algorithms, with less discussion of implementation issues and few nontrivial numerical results. This paper focuses on the parallel implementation of several iterative substructuring algorithms for elliptic partial differential equations in three dimensions. Full analysis of some of the algorithms, using standard domain decomposition techniques, can be found in Smith [21] and Dryja, Smith, and Widlund [12]. The underlying algorithm was first introduced in Smith [22].

Iterative substructuring algorithms are domain decomposition algorithms in which nonoverlapping subdomains are used. A preconditioned conjugate gradient method is used to solve the linear system obtained by a finite element discretization of the partial differential equation. The preconditioner is obtained by separately solving linear systems associated with the interiors of the subdomains, the faces between subdomains, and a coarse grid system that provides global coupling between the subdomains.

There is a fundamental difference in the nature of finite element solutions of elliptic problems in two and three dimensions. The formulation of the coarse grid system that works well in two dimensions results in poor convergence in three dimensions; see Smith [21]. Hence, new analysis and numerical experiments must be carried out for problems in three dimensions. Earlier theoretical work on the subject that has strongly influenced our work can be found in Bramble, Pasciak, and Schatz [5], Dryja [11], Dryja and Widlund [13], Mandel [20], and Smith [21]. For a modern treatment of iterative substructuring algorithms in three dimensions, see Dryja, Smith, and Widlund [12]. Other large-scale experimental work in domain decomposition is described in Bjørstad and Hvidsten [1], Bjørstad, Moe, and Skogen [2], Keyes and Gropp [16], [17], De Roeck [9], De Roeck and Le Tallec [10], Le Tallec, De Roeck, and Vidrascu [18], and Mandel [19].

Any good iterative substructuring algorithm with exact interior solvers can be reformulated to use approximate interior solvers. This approach has been analyzed with some success in Börgers [4] and Haase, Langer, and Meyer [15]. In this paper we

*Received by the editors August 7, 1991; accepted for publication (in revised form) March 26, 1992. This work 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.

[†]Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439-4844 (bsmith@mcs.anl.gov).

report on experiments in which both approximate and exact interior solvers are used. We use multigrid for the approximate interior solver in our experiments.

This paper is organized as follows. In §2 we introduce the algorithm using matrix notation. In §3 we discuss the implementation of the algorithm on distributed-memory machines. In §4, we present numerical results for some piecewise constant coefficient problems. In §5, we present an example from linear elasticity. In §6, we discuss future work, which will focus on the application of the algorithms to more difficult multicomponent elliptic partial differential equations.

2. Matrix form of preconditioner. Consider a scalar, second-order, self-adjoint, H^1-coercive, bilinear form $a_\Omega(u,v)$ on $\Omega \subset R^3$, and impose a homogeneous Dirichlet boundary condition on $\Gamma_0 \subset \partial\Omega$ and a Neumann boundary condition on $\partial\Omega \setminus \Gamma_0$. We assume that the underlying elliptic operator has no zero-order terms. Let $H^1_{\Gamma_0}(\Omega)$ be the subspace of functions in $H^1(\Omega)$ that vanish on Γ_0. The variational problem is to find $u \in H^1_{\Gamma_0}(\Omega)$ such that

$$a_\Omega(u,v) = (f,v) \quad \forall v \in H^1_{\Gamma_0}(\Omega).$$

We triangulate the domain Ω using the usual rules for finite element triangulations. Let $V^h(\Omega) \subset H^1_{\Gamma_0}(\Omega)$ be the space of continuous, piecewise linear functions on the triangulation that vanish on Γ_0. In addition, for the construction of the preconditioner, we assume that the set of elements is partitioned into disjoint substructures Ω_i. Let H be the characteristic diameter of the substructures; that is, assume that there exist constants c and C independent of h and H such that for all substructures, $cH \leq \operatorname{diam}(\Omega_i) \leq CH$. In the experiments reported here, the substructures are always logically brick-shaped, but this is not necessary for the algorithm.

The discrete problem is to find $u^h \in V^h(\Omega)$ such that

$$a_\Omega(u^h,v^h) = (f,v^h) \quad \forall v^h \in V^h(\Omega). \tag{1}$$

If u^h is expanded in the standard nodal basis $u^h = \sum_k u_k\phi_k$, the variational problem (1) can be written as the linear system

$$K\underline{u} = \underline{f}.$$

In previous work [21], we constructed preconditioners for this system that involve separately solving linear systems associated with the interiors of the subdomains, the faces shared by pairs of subdomains, and a system associated with the remaining degrees of freedom. The application of our preconditioner results in a convergence rate that is independent of the number of subdomains and is independent of jumps in the coefficients of the partial differential equation between subdomains.

We partition the unknown coefficients into those associated with the interiors of the subdomains, $\underline{u}_I$, those associated with the faces shared by exactly two subdomains, $\underline{u}_F$, and those shared by more than two subdomains (the wirebasket), $\underline{u}_W$. We use $\underline{u}_B$ to denote the vector of coefficients $(\underline{u}_F, \underline{u}_W)$. In addition, we let $\underline{u}^{(i)}$ represent the coefficients associated with the closure of subdomain Ω_i. The stiffness matrix K can be written in block form as

$$\begin{pmatrix} K_{II} & K_{IF} & K_{IW} \\ K_{IF}^T & K_{FF} & K_{FW} \\ K_{IW}^T & K_{FW}^T & K_{WW} \end{pmatrix}.$$

The Schur complement after we eliminate the interior unknowns is given by

$$\begin{pmatrix} S_{FF} & S_{FW} \\ S_{FW}^T & S_{WW} \end{pmatrix} = \begin{pmatrix} K_{FF} & K_{FW} \\ K_{FW}^T & K_{WW} \end{pmatrix} - \begin{pmatrix} K_{IF}^T \\ K_{IW}^T \end{pmatrix} K_{II}^{-1} \begin{pmatrix} K_{IF} & K_{IW} \end{pmatrix}.$$

We express the inverse of the stiffness matrix in partially factored form as

$$\begin{pmatrix} I & -K_{II}^{-1}K_{IF} & -K_{II}^{-1}K_{IW} \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & -S_{FF}^{-1}S_{FW} \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} K_{II}^{-1} & 0 & 0 \\ 0 & S_{FF}^{-1} & 0 \\ 0 & 0 & \tilde{S}_{WW}^{-1} \end{pmatrix}$$

$$\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & -S_{FW}^T S_{FF}^{-1} & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ -K_{IF}^T K_{II}^{-1} & I & 0 \\ -K_{IW}^T K_{II}^{-1} & 0 & I \end{pmatrix}.$$

The matrix S_{FF} represents that part of the Schur complement, after the interior nodes have been eliminated, that is associated with the coupling between the nodes on the faces of the subdomains. $\tilde{S}_{WW}$ is the Schur complement associated with the wirebasket once the unknowns of the interior and the faces have been eliminated. We do not explicitly form these matrices; instead, the preconditioner is constructed by replacing various blocks with more computationally attractive matrices. We note that K_{II}^{-1}, S_{FF}^{-1}, $S_{FF}^{-1}S_{FW}$, and $\tilde{S}_{WW}^{-1}$ may be replaced.

The K_{II} is a block diagonal matrix with a block for each subdomain interior. For K_{II}^{-1} we use either a sparse factorization (in particular, the Yale Sparse Matrix Package, which uses the minimum-degree algorithm to reorder to reduce fill-in) or at least one multigrid V-cycle to approximate the action of the inverse. In the latter case, the approximate inverse can be written as $\tilde{K}_{II}^{-1} = (I - M_\nu)K_{II}^{-1}$. M_ν is the symmetric error iteration matrix for ν multigrid V-cycles, and $\rho(M_\nu) < 1$. We note that there is considerable freedom in choosing the multigrid solver and the number of V-cycles. When exact interior solvers are used, we can eliminate the unknowns $\underline{u}_I$ initially and iterate on only $\underline{u}_B$, using one solver involving K_{II} per iteration. Once $\underline{u}_B$ is known, we can backsolve for $\underline{u}_I$. If an approximate solver is used for the interior, then two interior solvers are needed per iteration, and all of the variables are present in the iterative process.

We replace the Schur complement S_{FF} with a block diagonal matrix with one block for each face. Let $\tilde{S}_{FF}$ be the generic substitution. Several candidates exist for the matrix blocks. They are all derived by extending earlier results for problems in two dimensions. Unfortunately, no one substitution exists that is appropriate for all situations. Each has its own advantages and disadvantages in terms of computational expense and convergence properties. See §5 for an example where the need for a face preconditioner is avoided.

1. We can explicitly use the blocks from the Schur complement. The advantage of this approach is that the block is automatically well adapted for each differential equation. Unfortunately, it is very expensive to calculate the blocks, except for small subdomains. This method is probably practical only for extremely ill conditioned problems where no good substitutes exist.

2. We can use a suitable scaling of the J operator. The J operator is the square root of the two-dimensional discrete Laplacian on a regular, rectangular mesh. See Bramble, Pasciak, and Schatz [5] for a discussion of why this leads to good results. It has been shown that J is spectrally equivalent to the explicit block of the Schur complement. It is computationally cheap to apply the action of J^{-1} to a vector. It does not, however, adapt to the particular partial differential equation and for anisotropic problems can have very poor convergence properties.

3. For constant-coefficient elliptic partial differential equations on rectangular subdomains with a uniform finite difference mesh, we can exactly diagonalize the Schur complement associated with a face, using fast sine transforms. For two dimensions, Chan and Hou [6] have proposed the use of this fast spectral decomposition to approximate the actual Schur complement. This approach could be extended to three dimensions. Again, as with the J operator, this requires that we use a regular, rectangular mesh on brick-shaped subdomains.

4. We could use a multilevel preconditioner. This is an extension to three dimensions of the hierarchical Schur complement preconditioner considered in Smith and Widlund [23]; see also Haase, Langer, and Meyer [15]. This is not a spectrally equivalent preconditioner, but is nearly so. Like the J operator, it does not adapt to the particular partial differential equation.

5. Another approach is to use the tangential component of the original operator restricted to that face. It can be obtained easily and adapts reasonably well to the partial differential equation. This approach is taken in Chan and Keyes [7] and Keyes and Gropp [17]. It does not perform well, however, when the components of the operator that are normal to the face dominate.

6. The method of probing (see Chan and Keyes [7] and Chan and Mathew [8]) could be used to calculate diagonal or band diagonal approximations to the Schur complement on each face. This approach also does not result in a spectrally equivalent preconditioner; at best, the condition number grows like $(H/h)^{1/2}$.

We observe that the operator $S_{FF}^{-1}S_{FW}$ maps values from the boundaries of the faces to the faces. It is known that the most important property of the mapping is that it maps a constant value on the boundary of the face onto the face as the same constant. We use a simple mapping that preserves this property. We map the average of the unknowns on the boundary of the face onto the face; see Smith [21] for the underlying theory. Let T^T denote this mapping. More sophisticated interpolation schemes are also possible and may be needed for more difficult problems.

For $\tilde{S}_{WW}^{-1}$, inspired by Mandel [20], we use the matrix defined by the following minimization problem:

$$(2)\qquad \min_{\underline{u}} \sum \min_{\bar{w}^{(i)}} \delta_i(H/h)(\underline{u}_W^{(i)} - \bar{w}^{(i)}z^{(i)})^T I(\underline{u}_W^{(i)} - \bar{w}^{(i)}z^{(i)}) - \underline{u}_W^T f_W.$$

The $z^{(i)}$ is a vector of all ones of the same dimension as $\underline{u}_W^{(i)}$, and δ_i is a function of H/h given below. We let G_{WW} denote the matrix defined by the minimization given above. For nontrivial problems, a diagonal (or block diagonal) matrix that is adapted to the particular partial differential equation may be substituted in place of the identity matrix in the above formula. A simpler choice for this part of the preconditioner would be the block diagonal part of the original stiffness matrix associated with the wirebasket. This choice, however, results in a preconditioned system whose condition number grows faster than C/H^2, while the former choice results in a condition number bounded by $C(1 + \log(H/h))^2$; see Smith [21].

We write the generic form of the inverse of the preconditioner as

$$\begin{pmatrix} I & -\tilde{K}_{II}^{-1}K_{IF} & -\tilde{K}_{II}^{-1}K_{IW} \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & -T^T \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} \tilde{K}_{II}^{-1} & 0 & 0 \\ 0 & \tilde{S}_{FF}^{-1} & 0 \\ 0 & 0 & \tilde{S}_{WW}^{-1} \end{pmatrix}$$
$$\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & -T & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ -K_{IF}^T\tilde{K}_{II}^{-1} & I & 0 \\ -K_{IW}^T\tilde{K}_{II}^{-1} & 0 & I \end{pmatrix}.$$

We consider five specific preconditioners in the first set of numerical studies. The first involves diagonal preconditioning of the original stiffness matrix. We denote the preconditioner by D. The second and third preconditioners both use exact interior solvers. The first version involves solving the wirebasket problem with the technique introduced above and in Smith [21]. We can express this preconditioner as

$$B_G^{-1} = \begin{pmatrix} T^T \\ I \end{pmatrix} G_{WW}^{-1} \begin{pmatrix} T & I \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{S}_{FF}^{-1} \begin{pmatrix} I & 0 \end{pmatrix}.$$

In the second version, we solve the wirebasket problem using a diagonal matrix

$$B_D^{-1} = \begin{pmatrix} 0 \\ I \end{pmatrix} D_{WW}^{-1} \begin{pmatrix} 0 & I \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{S}_{FF}^{-1} \begin{pmatrix} I & 0 \end{pmatrix}.$$

The last two preconditioners use approximate solvers on the interior subproblems.

$$B_{GA}^{-1} = \begin{pmatrix} I & -\tilde{K}_{II}^{-1}K_{IB} \\ 0 & I \end{pmatrix} \begin{pmatrix} \tilde{K}_{II}^{-1} & 0 \\ 0 & B_G^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -K_{IB}^T\tilde{K}_{II}^{-1} & I \end{pmatrix},$$

$$B_{DA}^{-1} = \begin{pmatrix} I & -\tilde{K}_{II}^{-1}K_{IB} \\ 0 & I \end{pmatrix} \begin{pmatrix} \tilde{K}_{II}^{-1} & 0 \\ 0 & B_D^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -K_{IB}^T\tilde{K}_{II}^{-1} & I \end{pmatrix}.$$

For the piecewise constant coefficient problems considered in this paper, we use a multiple of the J operator as our face preconditioner. For our model problems, this approach works almost as well as the computationally more expensive explicit Schur complement.

3. Implementation issues. We have implemented the algorithm on a distributed-memory machine in which each processor has its own local memory and can communicate, either directly or indirectly, with all other processors by using explicit message passing. The specific architecture is the Intel iPSC/860. These machines have from 8 to 128 Intel i860 processors, each of which is capable of sustained rates of more than 4 megaflops with compiled Fortran or C. The peak performance for hand-coded assembler is somewhat higher. Each processor has between 8 and 16 megabytes of local memory. The Intel iPSC/860 machine has a hypercube connection between the nodes. For messages greater than 100 bytes in length, the startup time for nearest neighbor nodes is roughly 150 microseconds, while the peak transfer rate is 2.5 megabytes per second. These results were obtained experimentally by Bokhari [3].

Communication time on these machines is slow compared to the floating-point speed. Hence, data locality and the minimization of communication are vital.

The particular implementation of our algorithm is closely related to the work of Keyes and Gropp [16] for problems in two dimensions. Keyes and Gropp subdivide the domain into rectangular tiles. Each tile is then discretized by using a regular mesh. This is a very natural approach combining flexibility of the domain with regular subdomains and the possibility of local uniform mesh refinement.

The three-dimensional domain is partitioned into brick-shaped subdomains, each of which is assigned a uniform finite element or finite difference mesh. To simplify the coding, we require that adjacent subdomains share an entire face, entire edge, or vertex; see Fig. 1. This requirement is necessary for our implementation, but not for the underlying mathematical algorithms.

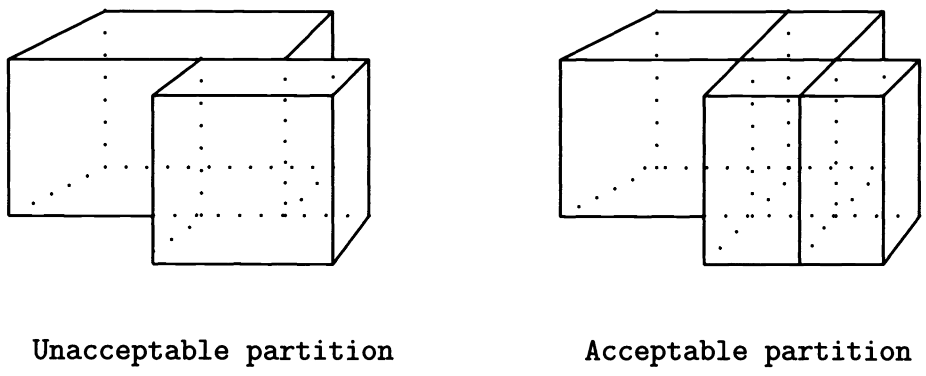


FIG. 1. *Unacceptable and acceptable partitions of a domain.*

Each processor is assigned one or more subdomains. The information pertaining to the interior of the subdomain is uniquely owned by that processor and is not directly available to any other processor. Each face, edge, or vertex is jointly owned by several subdomains and hence potentially by several processors. Because of this joint ownership, whenever a change is made to the part of the solution associated with a face, edge, or vertex of one subdomain, this information must be conveyed to the other joint owners by using explicit message passing. We refer to this process as *merging of partial data.* For each face, edge, and vertex, we designate one of the joint owners as the main owner and the others as auxiliary owners.

Essentially three types of communication between processors are required when the preconditioned conjugate gradient method is used to solve the linear system. The first is multiplication by the stiffness matrix. After the calculation of the local contribution to the matrix multiplication, the parts of the product vector that are shared by two or more processors must be merged. This merging of partial results can be performed in several ways. At this time a naive approach is used. The partial sums on each face, edge, and vertex are accumulated by the main owner and then sent out to the joint owners. For large problems, when using the full preconditioner, we find that less than five percent of the time is spent doing communication related to the matrix multiplication. With diagonal preconditioning, the matrix multiply dominates the entire solution time. Hence, optimizing the communication in the matrix multiply becomes important.

The application of the preconditioner is the most expensive operation in terms of communication. The principal reason is that the preconditioner is designed to provide for global communication of information in each step of the iteration process. When less communication is provided, more iterations are needed, although at a lower cost per iteration. With simple diagonal preconditioning, for instance, no cross-processor communication is needed. This fact suggests that for many well-conditioned problems diagonal scaling is the optimal approach for parallel computing systems of the type considered in this paper.

We list below the steps currently used in the application of the preconditioner B_G^{-1}. The steps in braces are the additional steps needed when approximate interior solvers are used.

1. {Approximate solvers on interior problems.}
2. {Merge results.}
3. Interpolate face averages onto the edges.
4. Merge results.
5. Calculate an average on the wirebasket for each subdomain, and send this to the coarse solver.
6. Solve the face problems and coarse problem simultaneously.
7. Interpolate the coarse solution to the wirebasket.
8. Merge results.
9. {Approximate solvers on interior problems.}

The conjugate gradient method requires several inner products per iteration. When possible, we use a direct call to a low-level implementation of a cross-processor inner product.

4. Experimental results for piecewise constant coefficient problems. In this section we report on experiments with scalar elliptic problems with piecewise constant coefficients. The reason for examining such problems is threefold: we can compare the well-developed theory with the numerical results, we can obtain a lower bound on how well the algorithm will perform for more difficult problems, and we can resolve questions about the optimal scaling of different parts of the preconditioner.

The total solution time depends on the number of iterations needed and the average amount of time needed per iteration. Information useful for comparing different algorithms is provided by the number of iterations needed to obtain a fixed accuracy of the solution. The square root of the condition number of the preconditioned system gives a bound on the number of iterations needed.

4.1. On the local bounds. In this algorithm, as with most iterative substructuring algorithms (see Dryja, Smith, and Widlund [12]), it is possible to bound the condition number of the preconditioned matrix by bounds obtained locally, that is,

$$\kappa(B_G^{-1}S) \le \frac{\max C_i}{\min c_i},$$

where the c_i and C_i satisfy

$$c_i \underline{u}^{(i)^T} B_G^{(i)} \underline{u}^{(i)} \le \underline{u}^{(i)^T} S^{(i)} \underline{u}^{(i)} \le C_i \underline{u}^{(i)^T} B_G^{(i)} \underline{u}^{(i)} \quad \forall \underline{u}^{(i)}.$$

We wish to determine how close the local bounds are to the actual condition numbers as a function of the number of subdomains. We have performed two sets of experiments, one using the exact blocks of the Schur complement, and the other using the

J operator as the face preconditioner. In both this and the following section, the domains are unit cubes. In all of the experiments we use the Lanczos algorithm to calculate the extreme eigenvalues from which we calculate the condition numbers. We make two observations from Table 1:

- The condition numbers when using either the explicit Schur complement or the J operator are virtually identical for the Laplace operator.
- The bounds obtained from the local analysis quite closely predict the condition numbers even for a relatively small number of subdomains.

The positions in the table denoted by a dash are cases for which experiments were not carried out because of time or memory constraints.

TABLE 1
Condition numbers and local bounds.

	Explicit Schur complement					J operator				
	Local	Number of subdomains				Local	Number of subdomains			
H/h	bound	27	64	125	216	bound	27	64	125	216
4	9.66	8.33	8.77	8.82	9.22	10.25	8.72	9.41	9.35	9.92
5	11.18	9.57	10.28	10.20	10.68	12.10	9.86	10.78	10.61	11.30
6	12.40	10.87	11.52	11.43	–	13.64	11.05	12.07	11.85	12.89
7	14.04	11.83	12.63	–	–	15.07	11.96	13.56	13.30	14.51
8	15.86	12.83	14.05	–	–	16.31	12.87	15.00	14.70	16.06
9	17.59	13.62	–	–	–	17.54	13.63	16.37	–	–
10	19.23	–	–	–	–	19.26	14.38	17.67	17.61	18.91
16	–	–	–	–	–	–	21.12	24.20	24.16	25.98
20	–	–	–	–	–	–	24.15	27.88	27.78	29.83

4.2. On the scaling of the coarse problem. We can express the preconditioned problem when using exact interior solvers as

$$B_G^{-1} = \begin{pmatrix} T^T \\ I \end{pmatrix} G_{WW}^{-1} \begin{pmatrix} T & I \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{S}_{FF}^{-1} \begin{pmatrix} I & 0 \end{pmatrix}.$$

The mathematical analysis of the algorithm (see Smith [21] and Dryja, Smith, and Widlund [12]) tells us that asymptotically, for large H/h, we should scale G_{WW} by a factor $\delta_i(H/h) = C(1+\log(H/h))$. The analysis gives no information, however, about the selection of the constant C nor whether the scaling is important for relatively small values of H/h. We shall refer to the case with $\delta_i(H/h) = 1$ as the natural scaling. In our experiments, we determine for each mesh size the optimal scaling $\delta_i(H/h)$ using a simple bisection method and compare the condition number with that obtained using the natural scaling. The results are presented in Table 2.

A related question is whether, in the construction of G_{WW} (see (2)), we should scale the diagonal elements for the nodes associated with the vertices of the subdomains differently from those nodes associated with the edges. The most natural choice is to scale the former elements by $\frac{1}{2}$, since those nodes are contained in exactly twice as many subdomains. We refer to the resulting choice as a weighted G_{WW}. We make the following conclusions from Table 2:

- For the range of computationally practical meshes on the subdomains (i.e., $H/h \leq 32$), the natural scaling is only trivially worse than the optimal scaling.
- Using the weighted G_{WW} results in only a trivial improvement in the condition number.

TABLE 2
Natural versus optimal scaling of coarse problem: condition numbers.

$G^{(i)}$	Scaling	H/h							
		4	8	12	16	20	24	28	32
Natural	Natural	9.4	15.0	20.1	24.2	27.9	31.1	33.9	36.5
	Optimal	8.8	14.0	19.3	23.8	27.6	30.9	33.9	36.5
Weighted	Natural	–	14.2	19.3	23.6	27.2	–	–	–
	Optimal	–	13.6	18.8	23.3	27.1	–	–	–

4.3. The growth of the condition numbers. Mathematical analysis predicts the growth in the condition number as a function of the mesh refinement H/h, and the number of subdomains, but it does not give good estimates of the actual numerical values. Our results are given in Tables 3 and 4. For the preconditioner labeled B_{G^A} we have used one multigrid V-cycle to solve the subdomain problems approximately. The difference in the condition number between using one multigrid V-cycle, two multigrid V-cycles, and an exact interior solver is very small. Similar results have previously been noted by Börgers [4] and Haase, Langer, and Meyer [15] for problems in two dimensions but they have not yet been fully explained theoretically.

TABLE 3
Growth in condition numbers for 64 *subdomains* ($H = 1/4$).

H/h	Unknowns	K	S	$B_G^{-1}S$	$B_{GA}^{-1}K$	$B_D^{-1}S$	$B_{DA}^{-1}K$
4	3,375	103	53.8	9.4	9.4	67.6	67.7
8	29,791	414	122	15.0	15.0	107	107
12	103,823	933	192	20.1	20.1	131	133
16	250,047	1,656	261	24.2	24.4	150	152
20	493,039	2,593	331	27.9	28.1	165	166
24	857,375	3,734	401	31.1	31.3	–	–
28	1,367,631	5,083	–	33.9	34.2	–	–
32	2,048,383	6,640	–	36.5	36.9	–	–
Observed growth		$(1/h)^2$	$1/(Hh)$	$(1+\log(H/h))^2$		$(1/H^2)(1+\log(H/h))^2$	

TABLE 4
Growth in condition numbers for 216 *subdomains* ($H = 1/6$).

H/h	Unknowns	K	S	$B_G^{-1}S$	$B_{GA}^{-1}K$	$B_D^{-1}S$	$B_{DA}^{-1}K$
4	12,167	232	119	9.9	9.9	155	149
8	103,823	933	269	16.1	16.1	230	232
12	357,911	2,099	421	21.5	21.5	281	283
16	857,375	3,734	573	26.0	26.1	318	321
20	1,685,159	5,835	726	29.8	30.0	336	350
Observed growth		$(1/h)^2$	$1/(Hh)$	$(1+\log(H/h))^2$		$(1/H^2)(1+\log(H/h))^2$	

4.4. A comparison with a Bramble, Pasciak, and Schatz algorithm. Since our basic algorithm is similar to one of the important algorithms introduced by Bramble, Pasciak, and Schatz [5], we have reproduced the experiments reported in their paper using the preconditioner B_G^{-1}. The first set of experiments is for a unit cube divided into eight subcubes. The stiffness matrix is derived from the usual finite difference discretization for the Laplace operator. The second problem is for a unit cube divided into 27 subcubes with a different constant coefficient on each subcube; see [5] for the values used. We see from Tables 5 and 6 that for this class of problem, the two preconditioners produce similar condition numbers. We note that these are relatively small problems and that diagonal scaling also works well.

TABLE 5
Comparison with BPS IV: *Laplacian operator.*

	Condition number			
H/h	Diagonal	BPS	$B_G^{-1}S$	Unknowns
4	25.3	13.9	10.3	343
8	103	17.7	12.9	3,375
16	414	23	18.4	29,791

TABLE 6
Comparison with BPS IV: *coefficients with jumps.*

	Condition number			
H/h	Diagonal	BPS	$B_G^{-1}S$	Unknowns
4	63.4	14.1	9.0	1,331
8	265.4	18.3	14.5	12,167

4.5. Timings. We next present timing results on a 32-node Intel iPSC/860 hypercube for a set of intentionally simple examples. We consider three problems. The first two problems are on the unit cube; the third is on a more complicated region depicted in Fig. 2. The unit cube is uniformly divided into subcubes Ω_{ijk}. In the third problem we use 244 subdomains which are not cubes; their aspect ratios are 4:5:20.

PROBLEM 1. Find u^h such that

$$\sum_{ijk} \int_{\Omega_{ijk}} \epsilon_{ijk}(\nabla u^h, \nabla v^h) = \int_\Omega f v^h \quad \forall v^h \in V^h.$$

The boundary conditions are given by $u^h = 0$ on $\partial\Omega$. The coefficients ϵ_{ijk} are constant on each subdomain and have large jumps between neighboring subdomains. Specifically, $\epsilon_{ijk} = \sin^2(16z)(e^{18x\sin(4y)} + e^{15(1-x)}) + 1$, where (x, y, z) is the center of Ω_{ijk}. The right-hand side is given by $f(x, y, z) = ze^x \sin(y)$.

PROBLEM 2. Find u^h such that

$$\int_\Omega (\nabla u^h, \nabla v^h) = \int_\Omega f v^h \quad \forall v^h \in V^h.$$

The solution u^h is constrained to be zero on one face of the cube and is free on the rest of the boundary. The right-hand side is the same as in Problem 1.

PROBLEM 3. This problem is the same as in Problem 2 except that the domain is as depicted in Fig. 2. The solution u^h is fixed on the bottom of the object and free on the rest of the object's boundary.

All the results are for one multigrid V-cycle sweep as an approximate solver for the interior problems, that is, two sweeps per subdomain per iteration. This choice was made because additional multigrid sweeps did not result in a decrease in the

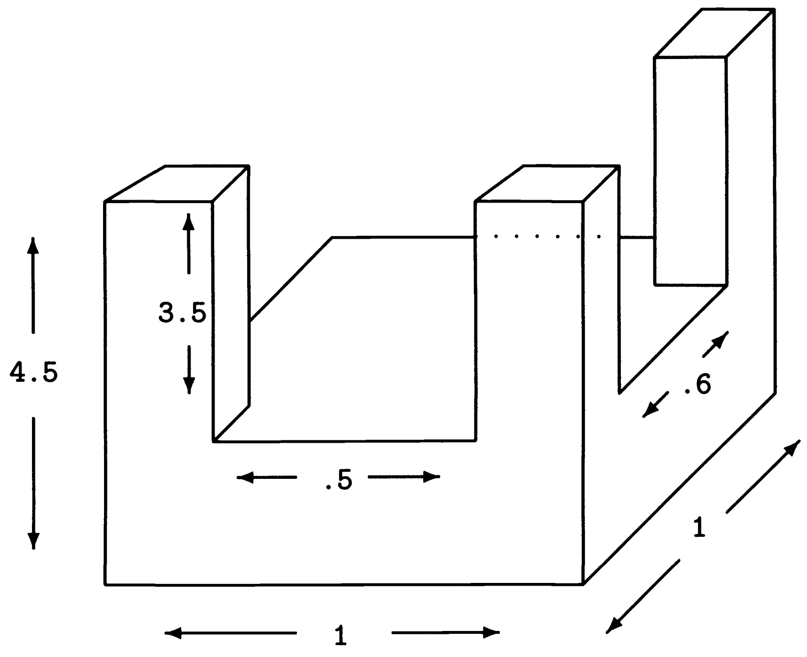


FIG. 2. *Domain for Problem* 3.

number of outer iterations. The times needed when exact interior solvers are used (i.e., with $B_G^{-1}S$) are much higher than those for the approximate solver. In addition, we cannot run the large problems when using exact interior solvers, as the sparse factors take a large percentage of the available space. For instance, for Problem 2 with 64 subdomains, the largest problem we could solve using the sparse interior solver was with a mesh of $H/h = 16$, while with multigrid we could solve problems with meshes up to $H/h = 32$. This fact suggests that for well-behaved problems like the Poisson problem, exact interior solvers, such as banded or sparse linear system solvers, are too expensive to be competitive. For more difficult problems, we do not yet know which approach is superior. We used the ordering routines in the Yale Sparse Matrix Package to order the unknowns for the sparse interior solvers. The nested dissection ordering might be a better choice.

The results for Problems 1 and 2 are given in Tables 7 and 8, respectively. We note that for Problem 2 both diagonal scaling and iterative substructuring without a coarse problem perform poorer than for Problem 1, while iterative substructuring performs essentially the same. This is due to the Neumann boundary conditions in Problem 2. The bounds on the convergence of iterative substructuring algorithms are independent of the ratio of Neumann boundary conditions to Dirichlet; see the comment in Smith [21]. On the other hand, the condition number of the original stiffness matrix deteriorates as one increases the percentage of the boundary with Neumann conditions.

For Problem 3 (see Table 9), the iteration counts are slightly higher than for Problem 2. It is well known that increasing aspect ratios cause a decay in the convergence rate of domain decomposition algorithms. In Problem 3 the algorithm with a diagonal scaling as the wirebasket problem performs poorly; see the column labeled

$B_{DA}^{-1}K$ in Table 9. This poor performance results because the condition number of the preconditioned problem grows like $1/H^2$, so the preconditioner becomes less effective when a large number of subdomains is used.

TABLE 7
Problem 1 with 64 subdomains (time in seconds).

H/h	Number of unknowns	Number of processors	Diagonal	$B_{GA}^{-1}K$	$B_{DA}^{-1}K$
8	29,791	Number of iterations	86	19	42
		Condition number	221	14.1	62
		4	14.1	14.8	23.0
		8	8.3	8.8	12.8
		16	5.7	5.4	8.0
		32	3.9	3.7	5.0
16	250,047	Number of iterations	169	24	49
		Condition number	885	23.1	89
		4	94.2	85.6	155.1
		8	51.7	44.9	80.9
		16	31.2	25.9	44.3
		32	17.0	14.3	23.2
20	493,039	Number of iterations	212	26	50
		Condition number	1,379	26.6	99
		8	113.0	109.4	187.7
		16	65.2	58.7	98.9
		32	34.2	30.4	50.7
24	857,375	Number of iterations	256	28	53
		Condition number	1,984	29.5	107
		8	193.7	163.5	290.5
		16	110.1	86.8	151.9
		32	57.4	44.8	77.6
32	2,048,383	Number of iterations	343	30	55
		Condition number	3,525	34.0	119
		32	153.9	119.3	207.3

4.6. Speed of computational kernels. Most large numerical codes have a few routines that perform the bulk of the numerical calculations and use most of the CPU time. We refer to these routines as the computational kernels. The best known computational kernels are the BLAS and FFT. The computational kernels involve no communication with other processes and should ideally vectorize and pipeline well. It is also important that they use the data and instruction caches well. Since the computational kernels dominate the time of the entire calculation, their optimization is important. On certain processors, the Intel i860, for example, the replacement of Fortran or C computational kernels with assembler language kernels can result in large decreases in the time of the calculation at the expense of a great deal of careful hand-coding of assembler code. We note that improvement in the speeds of computational kernels is a *local* optimization and does not involve communication or parallelization.

The present code is all written with standard Fortran and C computational kernels. We have observed the following floating-point speeds (see Fig. 3):

- Dot product: 4.9 million floating-point operations per second (MFLOPS).
- Matrix multiply: 7.8 MFLOPS.
- DAXPY: 4.8 MFLOPS.
- Multigrid solver: 3.4 MFLOPS.
- Diagonal preconditioner: 2.7 MFLOPS.

TABLE 8
Problem 2 *with* 64 *subdomains* (*time in seconds*).

H/h	Number of unknowns	Number of processors	Diagonal	$B_{GA}^{-1}K$	$B_{DA}^{-1}K$
8	34,848	Number of iterations	129	17	65
		Condition number	4,972	16.9	1,200
		4	24.1	16.6	40.7
		8	15.5	9.0	22.2
		16	8.5	5.5	13.0
		32	5.4	3.9	7.5
16	270,400	Number of iterations	262	23	78
		Condition number	19,916	27.2	1,644
		4	159.8	92.9	261.7
		8	86.9	48.2	136.1
		16	49.7	26.1	72.2
		32	26.6	14.1	37.6
20	524,880	Number of iterations	325	25	83
		Condition number	31,121	31.6	1,788
		8	168.0	89.9	265.3
		16	94.1	48.2	138.4
		32	49.8	25.3	71.4
24	903,264	Number of iterations	388	28	89
		Condition number	44,817	38.8	1,911
		8	304.2	170.3	481.6
		16	168.2	89.5	248.5
		32	87.6	46.8	129.1
32	2,130,048	Number of iterations	522	32	91
		Condition number	79,682	51.0	2,101
		32	233.4	130.8	347.8

TABLE 9
Problem 3 *with* 244 *subdomains* (*time in seconds*).

H/h	Number of unknowns	Number of processors	Diagonal	$B_{GA}^{-1}K$	$B_{DA}^{-1}K$
8	132,792	Number of iterations	309	20	152
		Condition number	19,657	20.8	4,710
		8	143.2	54.9	217.4
		16	78.2	35.9	114.9
		32	48.3	26.6	66.4
16	1,030,512	Number of iterations	617	35	179
		Condition number	78,486	74.1	6,428
		16	427.1	155.1	629.0
		32	237.2	88.5	332.1
20	2,000,460	Number of iterations	772	39	187
		Condition number	122,582	93.9	6,981
		32	453.4	157.5	622.7

- Sparse factorization: 3.8 MFLOPS.
- Sparse triangular solvers: 4.9 MFLOPS.

These results were obtained from the largest set of problems listed in Table 8. They do not fit completely in cache.

In the problem with 2,130,048 unknowns listed in Table 8, the per-processor flop rate for the entire calculation (from distributing the geometric information to the nodes, to solving the system) was 4.0 MFLOPS for the diagonal preconditioner and 3.1

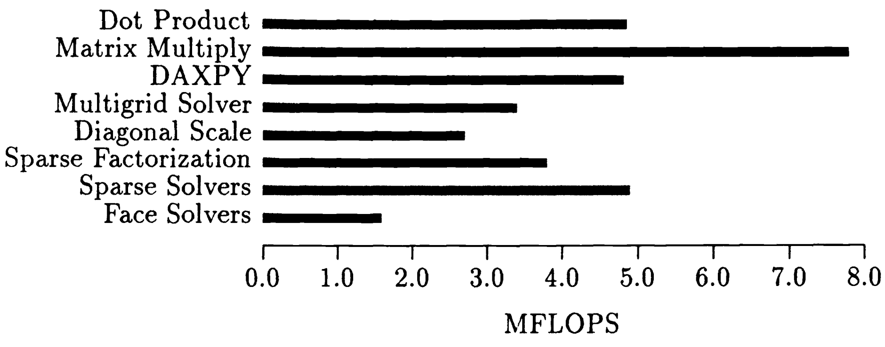


FIG. 3. *Flop rate in the computational kernels.*

MFLOPS for the more sophisticated preconditioner. Yet the diagonal preconditioner took more than twice as much time. The lower overall flop rate for the sophisticated preconditioner can be explained by the much lower flop rate of the multigrid solver.

The per-processor flop rate was obtained by taking the total number of floating-point operations performed on the processor and dividing by the total time the processor was in operation, including the time it was communicating with the other processors. While this number is a useful indicator of how well the processor is being utilized, it should not be overemphasized. The goal is to *minimize total computation time.* The best algorithm is the one that does exactly that, even if its per-processor flop rate is lower than that for other algorithms.

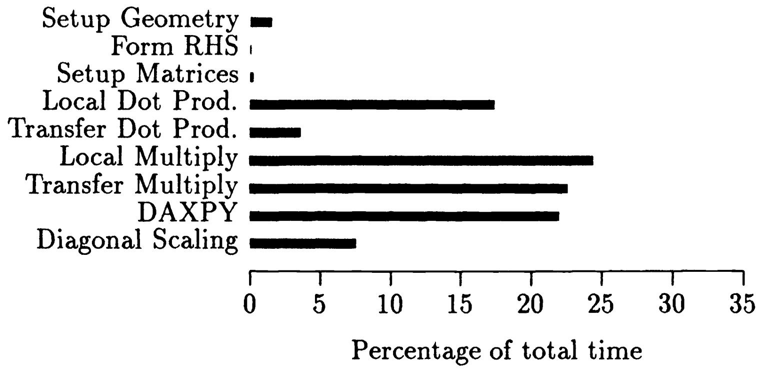


FIG. 4. *Percentage of time in different states, diagonal preconditioner.*

In Figs. 4 and 5, we graph the percentage of total wall-clock time spent in each portion of the code for the diagonally preconditioned and fully preconditioned problems. This gives a clear indication of what part of the code can most fruitfully be optimized. We also graph, in Fig. 6, the percentage of wall-clock time spent in various parts of the code for Problem 2 when the sparse interior solver is used. This is for 64 subdomains with a mesh of $H/h = 16$, the largest problem we could fit onto 32 processor nodes while using the sparse direct solver to solve the interior problems. The overall flop rate obtained here was 3.8 MFLOPS.

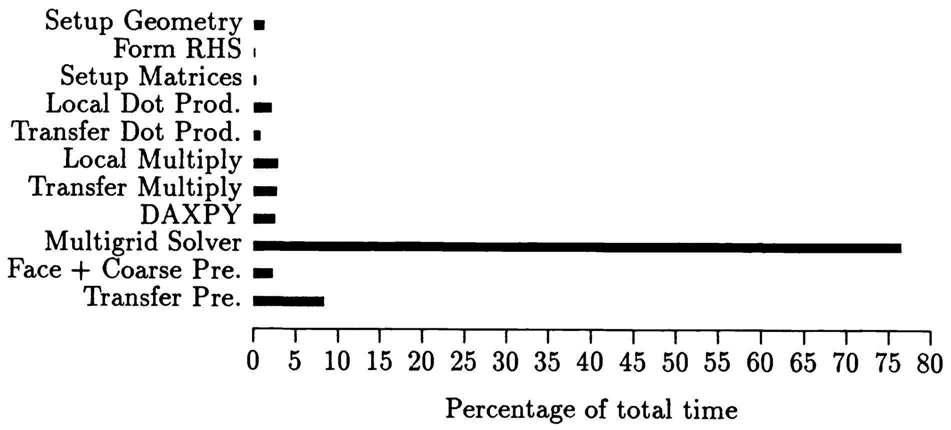


FIG. 5. *Percentage of time in different states, full preconditioner.*

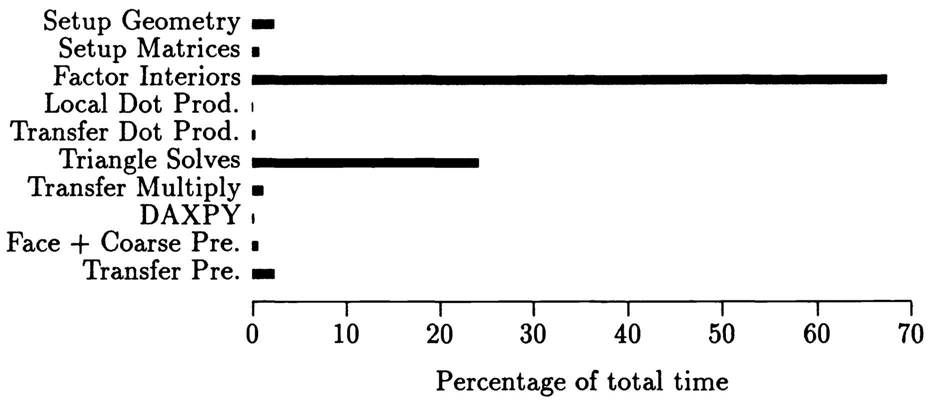


FIG. 6. *Percentage of time in different states, sparse solver.*

For the largest problem for which the sparse solver was used, 4.7 percent of the total computation time was spent on interprocessor communication. When multigrid was used to solve the interior problems approximately, the communication time increased to 11.7 percent of the total time. With diagonal preconditioning, 26.2 percent of the time was devoted to interprocessor communication.

5. Domain decomposition for elasticity. In this section we present numerical results for a minimal overlapping domain decomposition preconditioner applied to the equations of linear elasticity. The equations of linear elasticity are a self-adjoint coupled elliptic system that can be written as

$$\frac{E}{1+\nu}\left(\Delta \mathbf{u} + \frac{1-\nu}{1-2\nu}\nabla\nabla\cdot\mathbf{u}\right) = \mathbf{f}.$$

E is the Young's modulus, and ν is the Poisson ratio; both constants depend on the physical properties of the material being modeled. For our experiments we used

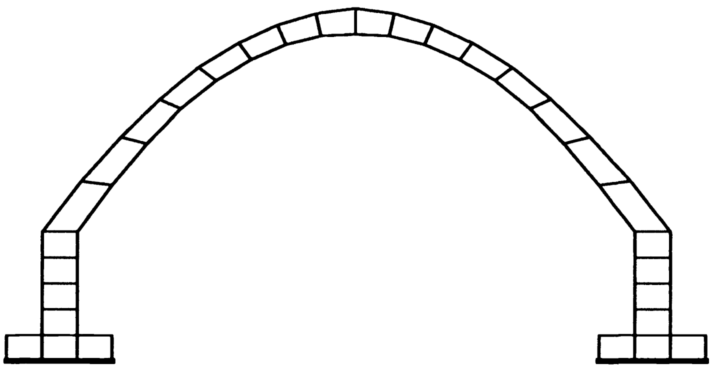


FIG. 7. *Arch domain.*

$E = 1$ and $\nu = 0.3$. The stopping criteria was a relative decrease in the l^2 norm of the residual of 10^{-5}.

We consider the domain as indicated in Fig. 7 and discretize the problem using 20-node, incomplete quadratic serendipity elements (cf. Zienkiewicz and Taylor [24]). We divide the domain into six subdomains. Since for the equations of linear elasticity we do not have a simple face preconditioner like the J operator, we use an overlap between domains of one mesh width; see Dryja and Widlund [14].

The stiffness matrix for the problem is extremely ill conditioned; for instance, even for a discretization with fewer than 1000 unknowns, the conjugate gradient method with diagonal scaling fails to converge even after 5000 iterations. The calculations given in Table 10 were carried out on a Sun Sparcstation 2. For this experiment we used exact interior solvers with a nested dissection ordering.

TABLE 10
Arch problem.

Number of unknowns	Without coarse system		With coarse system	
	Iterations	Condition	Iterations	Condition
2541	39	607	15	7.6

6. Conclusions and future directions. We believe that our results indicate that variants of the iterative substructuring approach are viable techniques for the solution of elliptic partial differential equations in three dimensions on modern distributed-memory machines. For model problems, the iterative substructuring algorithm performs better than diagonal scaling, but not by an enormous factor, indeed, not by enough to justify the extra burden imposed by coding the algorithms. However, we believe that for nontrivial problems, the difference between the two approaches in terms of computational time will increase. An example of this is given in §5, where diagonal scaling does not even converge, but the preconditioned problem requires fewer than 20 iterations.

We also note that for solving extremely large model problems, it is important to use approximate solvers for the interior problems. The fill from a band or sparse solver begins to dominate the memory usage, making it impossible to solve extremely large problems. For difficult problems, we may not be able to find an iterative solver for the interior problems that performs well enough to replace the direct solver, and this fact might limit the size of the problems that we can solve.

The iterative substructuring algorithm considered here works well on simple, piecewise, constant coefficient problems. To be useful in practice, it must be adaptable to a wide range of multicomponent elliptic partial differential equations. We therefore plan to focus on adapting each piece of the algorithm to a wide range of differential equations. The parts of the algorithm that must be generalized are the face preconditioners, the wirebasket coarse problem, and the interpolation onto the faces from the wirebasket. In addition, we shall consider other approaches to building interior iterative solvers, such as incomplete factorizations. Finally, we shall consider nonsymmetric problems.

Acknowledgments. I thank William Gropp for his excellent advice on the development of this implementation of the algorithm. In addition, I thank Olof Widlund for his comments on an early draft of this paper.

REFERENCES

[1] P. E. BJØRSTAD AND A. HVIDSTEN, *Iterative methods for substructured elasticity problems in structural analysis*, 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. 301–312.

[2] P. E. BJØRSTAD, R. MOE, AND M. SKOGEN, *Parallel domain decomposition and iterative refinement algorithms*, in Parallel Algorithms for PDEs, Proceedings of the 6th GAMM-Seminar held in Kiel, Germany, January 19–21, 1990, W. Hackbusch, ed., Vieweg-Verlag, Braunschweig, Wiesbaden, 1990.

[3] S. BOKHARI, *Communication overhead of the Intel iPSC/860 hypercube*, ICASE Interim Report 10, ICASE, Hampton, VA, May 1990.

[4] C. BÖRGERS, *The Neumann–Dirichlet domain decomposition method with inexact solvers on the subdomains*, Numer. Math., 55 (1989), pp. 123–136.

[5] 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.

[6] T. F. CHAN AND T. Y. HOU, *Eigendecomposition of domain decomposition interface operators for constant coefficient elliptic problems*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1471–1479.

[7] T. F. CHAN AND D. F. KEYES, *Interface preconditioning for domain-decomposed convection-diffusion operators*, Tech. Rep. CAM 89-28, Dept. of Mathematics, University of California, Los Angeles, CA, 1989.

[8] T. F. CHAN AND T. P. MATHEW, *The interface probing technique in domain decomposition*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 212–238.

[9] Y.-H. DE ROECK, *A local preconditioner in a domain-decomposed method*, Tech. Rep. TR89/10, Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France, 1989.

[10] Y. DE ROECK AND P. LE TALLEC, *Analysis and test of a local domain decomposition preconditioner*, in Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, Y. Kuznetsov, G. Meurant, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991, pp. 112–128.

[11] 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, PA, 1988, pp. 43–61.

[12] M. Dryja, B. F. Smith, and O. B. Widlund, *Schwarz analysis of iterative substructuring algorithms for problems in three dimensions*, Mathematics and Computer Science Division Preprint MCS-P250-0791, Argonne National Laboratory, Argonne, IL, 1991.

[13] M. Dryja and O. B. Widlund, *Some domain decomposition algorithms for elliptic problems*, in Iterative Methods for Large Linear Systems, Academic Press, San Diego, CA, 1989, pp. 273–291.

[14] M. Dryja and O. B. Widlund, *Domain decomposition algorithms with small overlap*, Tech. Rep., Courant Institute, New York, 1992.

[15] G. Haase, U. Langer, and A. Meyer, *A new approach to the Dirichlet domain decomposition method*, Tech. Rep., Technical University of Chemnitz, Chemnitz, Germany, 1990.

[16] 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.

[17] ———, *Domain decomposition techniques for nonsymmetric systems equations: Examples from computational fluid dynamics*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 312–339.

[18] P. Le Tallec, Y.-H. De Roeck, and M. Vidrascu, *Domain-decomposition methods for large linearly elliptic three dimensional problems*, Tech. Rep. TR/PA/90/20, Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, Toulouse, France, 1990.

[19] J. Mandel, *Hierarchical preconditioning and partial orthogonalization for the p-version finite element method*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 141–156.

[20] ———, *Two-level domain decomposition preconditioning for the p-version finite element version in three dimensions*, Internat. J. Numer. Methods Engrg., 29 (1990), pp. 1095–1108.

[21] B. F. Smith, *A domain decomposition algorithm for elliptic problems in three dimensions*, Numer. Math., 60 (1991), pp. 219–234.

[22] ———, *Domain decomposition algorithms for the partial differential equations of linear elasticity*, Ph.D. thesis, Courant Institute of Mathematical Sciences, September 1990; also Tech. Rep. 517, Dept. of Computer Science, Courant Institute, New York, 1990.

[23] 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.

[24] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, 4th ed., McGraw-Hill, London, 1989.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 424–445, March 1993

MODIFIED CHOLESKY FACTORIZATIONS FOR SPARSE PRECONDITIONERS*

TAMAR SCHLICK†

Abstract. In large-scale unconstrained optimization problems, preconditioned conjugate gradient techniques are often used to solve the Newton equations approximately. In certain applications, "natural" sparse preconditioners can be derived from the structure of the problem and can accelerate convergence significantly. Since these preconditioners are not necessarily positive definite, modified Cholesky (MC) factorizations can be applied to construct related positive definite preconditioners. This paper describes such an adaptation of two MC techniques–GMW (Gill–Murray–Wright) and SE (Schnabel–Eskow)–in a truncated Newton minimization method. This paper then analyzes their effects on three interesting test problems of moderate size. The preliminary results suggest that the two MC algorithms perform quite differently in practice. Trends can be noted for sparse problems that differ from the dense case. Differences in the size of the modifications and in their variances throughout the minimization are observed and related to problem structure and minimization progress. These differences suggest that, for the minimization method examined, SE may be advantageous for highly nonlinear functions, while GMW may be more effective for functions that are well approximated by locally convex quadratic models.

Key words. modified Cholesky factorizations, preconditioned conjugate gradient methods, sparse preconditioners, truncated Newton minimization

AMS(MOS) subject classifications. 49, 65, 92

1. Introduction.

In large-scale unconstrained optimization problems, preconditioned conjugate gradient (PCG) techniques are often used to solve the Newton equations approximately. This approximate solution of the Newton equations produces a search vector at every step. In this general framework, each iteration involves minimization of a quadratic model of the objective function $F(\mathbf{x})$, $\mathbf{x} \in \mathbf{R}^n$, around the current iterate $\mathbf{x}_k$. This approximation along a direction $\mathbf{p}$ is given by

$$F(\mathbf{x}_k + \mathbf{p}) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T\mathbf{p} + \tfrac{1}{2}\mathbf{p}^T H_k \mathbf{p}, \tag{1}$$

where $\mathbf{g}_k$ and H_k are the gradient vector and Hessian matrix, respectively, of F at $\mathbf{x}_k$. Minimization leads to the Newton system of equations:

$$H_k\mathbf{p} = -\mathbf{g}_k. \tag{2}$$

The next iterate is defined by $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{p}_k$, where $\mathbf{p}_k$ solves (2) approximately, and $\lambda_k > 0$ is determined in a line search algorithm to guarantee sufficient function decrease [3], [10].

Truncated Newton methods form a subclass of Newton methods that control the size of the residual of (2) systematically [2], [4], [15]–[18], [21], [22], [24], [26]. For large-scale problems, PCG methods are natural for solving (2). Indeed, with a good combination of preconditioner and truncation strategy, truncated Newton methods can exhibit rapid convergence in relation to "nontruncated" analogues and other unconstrained minimization algorithms [15], [21], [22], [24], [26].

* Received by the editors February 12, 1990; accepted for publication (in revised form) March 18, 1992. This work is made possible through generous support from the National Science Foundation (CHE-9002146 and Presidential Young Investigator award ASC-9157582), the Searle Scholars program, New York University (Faculty of Arts and Science), the San Diego Supercomputer Center, and the Academic Computing Facility at New York University.

† Courant Institute of Mathematical Sciences and Chemistry Department, New York University, 251 Mercer Street, New York, New York 10012.

"Natural" preconditioners arise in many optimization applications from the separability of the objective function. For example, in computational chemistry problems, a sparse preconditioner is suggested by the existence of short-range and long-range chemical interactions: The "local" energy components, in contrast to the "nonlocal" terms, are rapid to evaluate, tend to cluster near the diagonal, and are typically large in magnitude [23], [24]. In mathematical biology problems involving interaction of elastic structures and fluid dynamics (e.g., models of blood clotting), the local connectivity structure describing the forces also leads to natural sparse preconditioners [22]. In computational geometry applications involving crystal solidification (as in formation of ice in water), the energy describing the evolution of the liquid/solid phase boundary contains a surface term for the interface and a volume integral; the boundary term leads to a natural preconditioner since it is both sparse and stiff in comparison to the volume term [1].

When the preconditioner M is not sufficiently positive definite, modified Cholesky (MC) factorizations can be adapted for the minimization method to construct a related positive definite sparse preconditioner $\overline{M}$. While the idea of *indefinite* preconditioners may be unconventional for certain applications, we have found this strategy to work well in practice in applications as mentioned above. In the typical case, the "natural" preconditioner may be indefinite only in some early Newton iterations; it becomes positive definite at later iterations [22].

The choice of MC strategy may play a critical role in governing performance in this context. The Gill–Murray–Wright (GMW) [10] and Schnabel–Eskow (SE) [8], [25] MC schemes offer two approaches. Both methods augment the original matrix by a nonnegative diagonal matrix. They differ both in the numerical choices for the modifications and in the process by which diagonal modifications are applied. Here we report computational results from truncated Newton minimization involving the two MC techniques for modifying sparse preconditioners.

In §2 we briefly review the GMW and SE MC algorithms. In §3 we describe the use of the MC factorizations in our truncated Newton minimization algorithm. Numerical results are presented and analyzed in §4. We use three problems of moderate size, two from computational chemistry and one a familiar optimization test problem. We discuss the results alongside eigenvalue analysis for the corresponding preconditioners and Hessians. Our observations regarding the performance of the algorithms using the GMW and SE schemes are summarized in §5. Results suggest various trends of MC effects for sparse problems that differ from the dense case. Different overall variances in the size of the modifications are observed and related to problem structure and minimization progress.

2. MC factorizations.

MC techniques are used to solve linear systems $M\mathbf{z} = \mathbf{r}$ for coefficient matrices M that are symmetric but not necessarily positive definite. They are not intended to solve systems in the usual sense, since the modified system $\overline{M}\bar{\mathbf{z}} = \mathbf{r}$, with $\overline{M} \neq M$, may produce a solution $\bar{\mathbf{z}}$ that bears no resemblance to $\mathbf{z}$. They are appropriate, however, when justification exists for modifying a linear system, as in Newton-type methods for nonlinear optimization.

MC algorithms begin with a symmetric $n \times n$ matrix M and produce a factorization

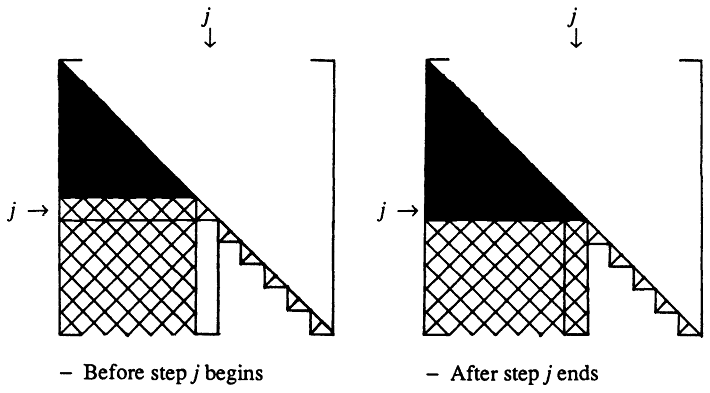
$$LDL^T = \overline{M} = M + E, \tag{3}$$

where L is unit lower-triangular, D is a positive diagonal matrix, $\overline{M}$ is positive definite, and E is a nonnegative diagonal matrix, at similar cost to a standard Cholesky factorization. The MC algorithm of GMW [10] has been used extensively in nonlinear optimization, exhibiting very good performance. The more recent MC scheme of SE [25] presents an alternative. Since it is impossible to define the "best" choice of $\overline{M}$ for an indefinite system, the two MC algorithms attempt to balance several practical and theoretical issues. Both attempt to combine the objectives of keeping $\|E\|$ small and producing a well-conditioned $\overline{M}$. That is, by balancing the amounts added to the diagonal at earlier and later iterations of the factorization, the modification strategy attempts to balance: (1) on one extreme, selecting large modifications that guarantee positive-definiteness but perturb the original matrix excessively; with (2) on the other extreme, choosing small, just sufficient additions that may lead to very large modifications later. In comparison to GMW, the SE procedure has: similar cost (involving an additional multiple of n^2 operations to the standard Cholesky factorization); a lower a priori upper bound on $\|E\|_\infty$; and, in many instances, a smaller value for $\|E\|_\infty$ [25].

The two procedures (without interchanges) are summarized below; see [10], [8], and [25] for details. The inclusion of interchanges does not alter the upper bounds on $\|E\|_\infty$, although it tends to produce better results in practice. Consider a single step of a row-wise MC factorization without interchanges. At the beginning of step j, $j = 1, \ldots, n$, final values are in place for the first $j-1$ elements of D and the first $j-1$ rows of L (see Fig. 1). The space originally occupied by M is overwritten while performing the factorization, and all subscripted quantities actually refer to locations in the M array. Here we treat portions of this array as two conceptual matrices: a diagonal matrix $\tilde{D}$ and an auxiliary matrix C. $\tilde{D}$ is initialized to the diagonal of M; element j of $\tilde{D}$ may be viewed as a trial value for d_j. The matrix C is stored in the first $j-1$ columns of rows j through n. The following calculations are performed during step j.

1. Compute row j of L:

$$\ell_{js} = c_{js}/d_s, \qquad s = 1, \ldots, j-1. \tag{4}$$



■ contains final L and D factors

⊠ contains the auxiliary quantities of LD

◺ contains updated diagonal elements

FIG. 1. *Step j of the MC factorization.*

2. Compute $\mathbf{c}_j = (c_{j+1,j}, c_{j+2,j}, \ldots, c_{n,j})^T$ (column j of C):

(5) $$c_{sj} = m_{sj} - \sum_{k=1}^{j-1} \ell_{jk} c_{sk}, \qquad s = j+1, \ldots, n.$$

Calculate the nonnegative quantity θ_j from $\mathbf{c}_j$:

(6) $$\theta_j = f(\mathbf{c}_j),$$

where the function f varies with the MC algorithm (see below).

3. Use θ_j, as specified by the MC algorithm, to compute $e_j \geq 0$. Set the final value of d_j to

(7) $$d_j = \tilde{d}_j + e_j.$$

We refer to e_j as the *modification*; when $e_j > 0$, we say that d_j has been *modified*.

4. Update $\tilde{D}$:

(8) $$\tilde{d}_s = \tilde{d}_s - c_{sj}^2 / d_j, \qquad s = j+1, \ldots, n.$$

The details of steps 2–4 above will now be discussed in turn for the two factorizations.

2.1. The GMW factorization. In GMW, the values for e_j are chosen so as to minimize an a priori upper bound on $\|E\|_\infty$ subject to the condition that positive definite matrices will not be perturbed (i.e., M positive definite $\Rightarrow E = 0$). Two numbers are required to perform the algorithm: ϵ_m (machine precision) and τ, the smallest acceptable value for any element of D, often $\sqrt{\epsilon_m}$ or $\sqrt[3]{\epsilon_m}$. Three other quantities are computed before the factorization begins:

(9) $$\gamma = \max_j \{ |m_{jj}| \},$$

(10) $$\xi = \max_{j>s} \{ |m_{js}| \},$$

and the bound

(11) $$\beta = \max \left\{ \gamma, \frac{\xi}{\max\{1, \sqrt{n-1}\}}, \epsilon_m \right\}.$$

In step 2, the GMW definition of θ is

(12) $$\theta_j = \|\mathbf{c}_j\|_\infty = \max\{|c_{sj}|\}, \quad s = j+1, \ldots, n.$$

In step 3, d_j is given by

(13) $$d_j = \max\{|\tilde{d}_j|, \tau, \theta^2/\beta\},$$

which defines e_j implicitly.

Note here the following properties: (1) The matrix M must be assembled a priori so that γ and ξ can be computed. (2) It follows from (13) that $\tilde{d}_j$ may be modified even when it is positive and greater than the lower bound τ; this situation arises when M is not "sufficiently" positive definite. When $\tilde{d}_j$ is negative but exceeds both τ and θ^2/β in magnitude, the final d_j is taken as $|\tilde{d}_j|$, which corresponds to a modification $e_j = 2|\tilde{d}_j|$. This result often occurs when M has at least one nonnegligible negative eigenvalue. (3) The restriction that $d_j \geq \theta^2/\beta$ implies that

(14) $$d_s \|\ell_{sj}^2\|_\infty \leq \beta;$$

hence the name "bound" for β.

2.2. The SE factorization. In SE, an attempt is made to compute the standard Cholesky factorization ("Phase 1") but include a special *phase test* that indicates when the smallest element of $\tilde{D}$ has become "too small." Once the phase test becomes true at any step, the SE modification rule is applied at all subsequent steps.

Two quantities must be specified: τ_1 and τ_2, typically of order $\sqrt[3]{\epsilon_m}$. The quantity γ (the largest magnitude of any diagonal of M) is defined by (9) as in GMW. At step j, let $d_{\min}$ denote $\min\{\tilde{d}_j, \ldots, \tilde{d}_n\}$. If

$$d_{\min} < \tau_1 \gamma, \tag{15}$$

the SE procedure enters Phase 2 and thereafter modifies diagonal elements according to the following rules. The value of θ_j (step 2) is the one-norm of $\mathbf{c}_j$:

$$\theta_j = \sum_{s=j+1}^{n} |c_{sj}| = \|\mathbf{c}_j\|, \tag{16}$$

so that θ_j is the current Gerschgorin radius for row j [11]. The modification for e_j, $1 \le j \le n-2$, is specified as

$$e_j = \max\{0, -\tilde{d}_j + \max\{\theta, \tau_2\gamma\}, e_{j-1}\}. \tag{17}$$

For steps $n-1$ and n, the SE procedure uses the explicit eigenvalues of the remaining 2×2 submatrix. At the end of step $n-2$, let $\lambda_{\ell o}$ and λ_{hi} denote these eigenvalues, with $\lambda_{\ell o} \le \lambda_{hi}$. Then e_{n-1} and e_n are both defined as

$$e_{n-1} = e_n = \max\left\{0, e_{n-2}, -\lambda_{\ell_0} + \tau_2 * \max\left\{\frac{1}{1-\tau_2}(\lambda_{hi} - \lambda_{\ell o}), \gamma\right\}\right\}. \tag{18}$$

The SE strategy produces a value for $\|E\|_\infty$ that is bounded above by the magnitude of the most negative lower Gerschgorin bound of the unfactorized matrix when Phase 2 begins. It also leads to a smaller a priori upper bound for $\|E\|_\infty$ than the GMW approach [25]. For GMW, the upper bound is roughly n^2 times the maximum element in M, whereas for SE it is roughly $2n$ times the maximum element in M.

3. MC factorizations in truncated Newton optimization.

MC factorizations are useful in nonlinear optimization for modifying the Hessian or some approximation to it involved in solving the Newton equation (2) for the search direction $\mathbf{p}$. Recall that truncated Newton methods solve the Newton equations only approximately, usually by some version of the conjugate gradient method. The rationale for the approximation is that global convergence can be achieved if the search direction is bounded away from orthogonality to the negative gradient; thus, it is never "necessary" to solve (2) accurately for global convergence. The size of the residual $\mathbf{r}_k = H_k\mathbf{p} + \mathbf{g}_k$ is controlled systematically at every step, and asymptotic quadratic convergence can be achieved if $\mathbf{r}_k$ is systematically made sufficiently small near the solution [2].

Truncated Newton methods are generally competitive with other large-scale optimization methods only when preconditioning is applied to the Newton equations [21], [26]. Fortunately, many applications in computational chemistry, geometry, and meteorology produce sparse preconditioners naturally from the "physics" of the problem. Often, such a "natural" preconditioner M can be derived from the separability of the objective function. It provides an approximation to H but may not be positive definite. An MC factorization can therefore be applied to produce a modified preconditioner, $\overline{M}$, that is positive definite.

We have adapted a truncated Newton algorithm targeted to sparse but not necessarily structured preconditioners [21], [22]. The linear system involving the preconditioner, $M\mathbf{z} = \mathbf{r}$, is solved by a *sparse* MC procedure based on the Yale Sparse Matrix Package [5]–[7]. Instead of a vector $\mathbf{z} = M^{-1}\mathbf{r}$ used in standard PCG algorithms, we obtain the MC solution $(\overline{M})^{-1}\mathbf{r}$. Our algorithm "TN," as used in the tests reported here, is described below. (Other variations have been considered; see [21] and [22] for details.) In this version, we assume that the sparsity structure of the preconditioner remains constant. This is a common situation in computational chemistry applications, since structure depends on the molecular connectivity, specified a priori.

3.1. Algorithm TN1: Outer loop of the truncated Newton method.

1. *Initialization*
   - Set $k = 0$ and evaluate $F(\mathbf{x}_k)$ and $\mathbf{g}_k$.
   - If $\|\mathbf{g}_k\| \le 10^{-8}\max\{1, \|\mathbf{x}_k\|\}$, where $\|\cdot\|$ is the standard Euclidean norm, exit algorithm. Otherwise, continue to step 2.
2. *Preparation for sparse* MC *factorization*
   - Determine the *pattern* of the preconditioner M. Since the upper triangle of M is stored in a compressed row format, the pattern is specified by two integer arrays that serve as row and column pointers [21].
   - Compute the permutation matrix P that minimizes *fill-in* (that is, produces a sparse Cholesky factor L for PMP^T, assuming that M is positive definite) by the minimum-degree reordering algorithm [5]–[7].
   - Compute the *symbolic factorization* of M. This involves determining the sparsity structure of the Cholesky factor L and preparing the corresponding data structure.
   - Evaluate the preconditioner and, optionally, the Hessian[1] (M_k and H_k, respectively).
3. (**Algorithm TN2**) *Inner loop*

   Compute a search vector $\mathbf{p}_k$ by solving $H_k\mathbf{p} = -\mathbf{g}_k$ approximately using a PCG method; see below.
4. *Line search*
   - Compute a step length λ by safeguarded cubic and quadratic interpolation [3], [14], so that $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda\mathbf{p}_k$ will satisfy

$$F(\mathbf{x}_{k+1}) \le F(\mathbf{x}_k) + \alpha\lambda\mathbf{g}_k^T\mathbf{p}_k, \tag{19}$$

$$|\mathbf{g}_{k+1}^T\mathbf{p}_k| \le \beta|\mathbf{g}_k^T\mathbf{p}_k|, \tag{20}$$

where $\alpha = 10^{-4}$ and $\beta = 0.9$.

[1] The Hessian is used only for the Hessian/vector products, $H\mathbf{d}$, in each PCG iteration of the inner loop (see Algorithm TN2). In some situations (e.g., H is sparse and structured), $H\mathbf{d}$ can be computed without explicit assembly of H. In many large-scale applications, H is dense and prohibitively expensive to calculate, but $H\mathbf{d}$ can be computed with *one* additional gradient evaluation by the finite-difference approximation $H\mathbf{d} \cong [\mathbf{g}(\mathbf{x}_k + \Delta x\,\mathbf{d}) - \mathbf{g}(\mathbf{x}_k)]/\Delta x$ where Δx is a suitably chosen interval, such as $\Delta x = 2\sqrt{\epsilon_m}(1 + \|\mathbf{x}_k\|)/\|\mathbf{d}\|$ [9], [21]. The user specifies the method of choice for the calculation of $H\mathbf{d}$.

- Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda \mathbf{p}_k$ and $k \leftarrow k+1$. (The new function value and gradient are known after the line search.)

5. *Convergence tests*
   If $\|\mathbf{g}_k\| \leq 10^{-8} \max\{1, \|\mathbf{x}_k\|\}$, exit algorithm. Otherwise, continue to step 6.
6. *Preparation for next Newton step*
   - Compute the preconditioner and, optionally, the Hessian (see previous footnote 1) at the new point (M_k and H_k, respectively).
   - Perform a "symmetric reordering" of M_k according to the permutation P. This only involves internal rearrangement of M.
   - Go to step 3.

3.2. Algorithm TN2: Inner loop k of the truncated Newton method. The sequence $\{\tilde{\mathbf{p}}_i\}$ below represents the PCG vectors used to construct $\mathbf{p}_k$ in step 3 of Algorithm TN1. We omit the Newton subscripts k from $\mathbf{p}$, $\mathbf{g}$, H, and M for clarity.

1. *Initialization*
   - Set $i = 0$, $\tilde{\mathbf{p}}_i = 0$, and $\mathbf{r}_i = -\mathbf{g}$.
   - Set the parameter η_k controlling the accuracy of the computed search vector:

$$\eta_k = \min\{\delta/k, \|\mathbf{g}\|\}, \tag{21}$$

   where $\delta \leq 1$.
2. *Preparation for sparse* MC *factorization*
   - Perform the *numerical factorization* of (the permuted) M by the chosen MC scheme. The resulting factors L and D of $P\overline{M}P^T = PMP^T + E = LDL^T$ are stored in the same sparse row format used for M. We refer to $\overline{M}$ as the "effective preconditioner."
   - Solve for $\mathbf{z}_i$ in

$$\overline{M}\mathbf{z}_i = \mathbf{r}_i \tag{22}$$

   by using the Cholesky factors in the system $(P\overline{M}P^T)P\mathbf{z}_i = P\mathbf{r}_i$:

$$L\mathbf{x} = P\mathbf{r}_i, \quad L^T\mathbf{y} = D^{-1}\mathbf{x}, \quad \mathbf{z}_i = P^T\mathbf{y}. \tag{23}$$

   - Set $\mathbf{d}_i = \mathbf{z}_i$.
3. *Negative curvature test*
   - Compute the matrix–vector product $\mathbf{q}_i = H\mathbf{d}_i$.
   - *If*

$$\mathbf{d}_i^T\mathbf{q}_i \leq \zeta(\mathbf{d}_i^T\mathbf{d}_i) \tag{24}$$

   (where ζ is a small number such as $\epsilon_m/100$)
   exit inner loop with search direction

$$\mathbf{p} = \begin{cases} \mathbf{z}_i & \text{if } i = 0, \\ \tilde{\mathbf{p}}_i & \text{else;} \end{cases} \tag{25}$$

   else continue to step 4.
4. *Truncation test*
   - Update the quantities

$$\begin{aligned} \alpha_i &= \mathbf{r}_i^T\mathbf{z}_i/\mathbf{d}_i^T\mathbf{q}_i, \\ \tilde{\mathbf{p}}_{i+1} &= \tilde{\mathbf{p}}_i + \alpha_i\mathbf{d}_i, \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \alpha_i\mathbf{q}_i. \end{aligned}$$

- *If*

$$\|\mathbf{r}_{i+1}\| \leq \eta_k \|\mathbf{g}\|, \tag{26}$$

exit inner loop with search direction

$$\mathbf{p} = \tilde{\mathbf{p}}_{i+1}; \tag{27}$$

else continue to step 5.

5. *Continuation of* PCG
   - Compute $\mathbf{z}_{i+1}$ as in step 2 (23) by reusing the MC factors of (the permuted) $\overline{M}$ so that

$$\overline{M}\mathbf{z}_{i+1} = \mathbf{r}_{i+1}. \tag{28}$$

   - Update the quantities

$$\begin{aligned} \beta_i &= \mathbf{r}_{i+1}^T \mathbf{z}_{i+1} / \mathbf{r}_i^T \mathbf{z}_i, \\ \mathbf{d}_{i+1} &= \mathbf{z}_{i+1} + \beta_i \mathbf{d}_i, \\ i &\leftarrow i + 1. \end{aligned}$$

   - Go to step 3.

3.3. Remarks on TN. In the negative curvature test of (24), we use the value $\zeta = 10^{-10}$. For the truncation test of (26), we use $\delta = 1$ in (21). (For diagnostic purposes, we also use $\delta = 10^{-8}$ in some runs to mimic a "nontruncated" version.) Thus the inner PCG loop can terminate if a direction $\mathbf{d}$ of negative curvature is detected, or if the truncation test is satisfied (see steps 3 and 4 above). In the first case, the resulting search direction (for the outer loop) is $-(\overline{M}_k)^{-1}\mathbf{g}_k$ if this occurs at the first PCG iteration, or the current PCG iterate, $\tilde{\mathbf{p}}_i$, otherwise. In the second case, the relative residual of the Newton equations $\tilde{r}_k = \|\mathbf{r}_k\|/\|\mathbf{g}_k\| \leq \eta_k < 1$ and $\lim_{k\to\infty} \sup(\tilde{r}_k/\|\mathbf{g}_k\|) < \infty$, which can be used to produce asymptotic quadratic convergence for the method [2]. Weaker, superlinear convergence can be obtained if the *forcing sequence* η_k is chosen so that $\lim_{k\to\infty} \eta_k = 0$ and $\tilde{r}_k \leq \eta_k < 1$ [2], [4]. Although we have found the residual test of (21) and (26) to work very well in practice, several alternatives have been suggested to produce an affine invariant truncation [4] or to avoid an excessive number of inner iterations [17], possible, for example, when $\|\mathbf{g}\|$ is small or H is ill conditioned.

Since the sparsity structure of the preconditioner remains constant throughout minimization, the permutation P is determined at the beginning of the algorithm. A no-pivoting MC algorithm is used to retain this structure. The symbolic factorization of M is also performed only once. In each inner loop, one numerical factorization is performed, whereas in each inner PCG iteration numerical solutions are required to solve $M\mathbf{z} = \mathbf{r}$ repeatedly for different right-hand side vectors ((22) for $i = 0$ and (28) for $i \geq 0$).

3.4. MC variations. In addition to the no-pivoting implementation of the MC factorization, our experience has suggested a variation of the SE formula in (17) and (18) for choosing the modification e_j. This involves omitting: (1) the condition $e_j \geq e_{j-1}$ in (17), and (2) the explicit eigenvalue strategy for e_{n-1} and e_n in (18). While $\|E\|_\infty$ may indeed be smaller if the original formulas are used, the special treatment of the last two variables may destroy inherent symmetry of the problem; furthermore, only small improvement on $\|E\|_\infty$ can generally be expected for large-scale systems. The *deletion* of the requirement that e_j be at least as large as e_{j-1} may be beneficial in our preconditioning context since fewer overall modifications to M may be involved. Moreover, for very large sparse systems, the rationale behind the original strategy (that a *larger* d_j may lead to a less negative trial value $\tilde{d}_{j+1}$ and hence a *smaller* modification e_{j+1}) may be irrelevant if $\ell_{j+1,j}$ or $c_{j+1,j}$ is zero (see (8)). In all our SE MC implementations, we omit the special strategy for determining e_n and e_{n-1}. We refer to versions "SE(a)" and "SE(b)" as the SE strategies where the formulas of (17) and

$$e_j = \max\{0, -\tilde{d}_j + \max\{\theta, \tau_2\gamma\}\}, \quad j = 1, \ldots, n, \tag{29}$$

are used, respectively.

4. Numerical experiments.

The following numerical examples are intended to suggest various trends that may occur in practice in our truncated Newton utilization of the various MC factorizations. Clearly, the special adaptation of MC for TN restricts any conclusions to the present context. The notion of "optimality" for one MC strategy versus another is very difficult because the *overall* effect on minimization is cumulative; only a thorough investigation on a wide variety of problems can attempt to address this question. However, our current experience may already prove useful, and we share it in the hope that further investigations will be motivated.

Three types of objective functions are examined, with variations of size, starting points, parameters, and preconditioners. The problems are chosen of moderate size to permit a thorough investigation of the MC segment, including eigenvalue analysis for H and M at every Newton iteration. Two functions are chosen from computational chemistry: one describing a typical case of a molecule with few and well-separated, low-energy conformational regions; and one of a cluster of molecules, for which many energetically favorable configurations exist. The third is a well-known, unconstrained minimization test function, sometimes called "trigonometric" [12], [13]; its Hessian is indefinite at the standard starting point, with many negative and clustered eigenvalues, and typically remains indefinite for many iterations. In general, many "standard" test problems are inappropriate for our comparison since natural, much less indefinite, nondiagonal preconditioners are nontrivial to formulate. For the trigonometric function, the Hessian is dense, so there is some justification for formulating various band preconditioners, which are also indefinite (see § 4.4).

4.1. Measures of overall performance.

Overall performance with the truncated Newton method can be measured by the total number of outer (Newton) iterations, inner (PCG) iterations (i.e., the sum over all Newton steps), and function and gradient evaluations. These quantities determine the computing time. Since function and gradient values are evaluated as a pair in the line search, we refer to this measurement as a "function evaluation."

The number of Newton iterations provides information on overall progress. Many Newton iterations, for example, typically indicate that indefiniteness of H has been detected, in which case the inner loop is terminated without the satisfaction of the truncation criterion. The number of PCG iterations has been suggested as a good overall measure of truncated Newton performance [2]. The number of function evaluations—at least one per Newton step and more if required by the line search to satisfy (19) and (20)—can be a good indicator of time for large-scale functions if function and gradient calculations dominate over the cost involved in Hessian and preconditioner calculations.[2]

An important consequence of the truncated Newton approach is that different algorithmic variations (e.g., truncation parameters, MC strategies) generate a different sequence of search vectors. This effect can lead to different overall performance. The effect of different MC procedures on overall minimization in our method will therefore be indirect and cumulative. While the relation between the MC strategy and the required number of PCG iterations to solve *a* given Newton system to completion is easier to measure, it is of more practical importance to understand the MC effect on overall minimization progress. This overall consideration is also relevant since in many large-scale problems, the number of PCG iterations is often limited to a number $\ll n$ due to the high cost of the inner iteration [16]. This implies that different MC strategies may lead to very different search directions from the same iterate. The total number of PCG iterations is both a measure of overall progress and a cumulative indicator of the step-by-step MC effect.

All computations were performed in double precision on a DEC Vax-3600 computer at the Courant Institute of Mathematical Sciences, New York University. Machine precision is of order 10^{-17}. We refer to the three MC strategies as: GMW (§2.1), SE(a), and SE(b) (§2.2 with variations indicated above). The MC tolerance parameters are: $\tau_1 = \tau_2 = 10^{-5}$ for SE, and $\tau = 10^{-8} * \max\{1, \gamma, \xi\}$ for GMW. This value of τ for GMW is chosen since our typical largest diagonal of M, γ in the computational chemistry problems has a value of $O(10^3)$ and satisfies $\gamma > \xi$.

4.2. A nucleic acid component. In "molecular mechanics," a potential energy function of atomic positions is minimized to predict three-dimensional structures of biomolecules for a given composition (e.g., base sequence for DNA) [19], [23]. Thus the independent variables are the Cartesian coordinates of all atoms in the system, denoted collectively by the vector $\mathbf{x}$. Potential energy surfaces are typically complex, involving many local minima, maxima, and transition points. Since the energy naturally decomposes into local and nonlocal interactions, the local components (among atoms in bonded sequences of atom pairs, triplets, and quadruplets) provide a natural sparse preconditioner which grows linearly in size with n [22]. We have found this choice of preconditioner ("M_{PE}") to work very well in practice in relation to other preconditioners and to no-preconditioning versions [22], [24].

Deoxycytidine (dC), is a component of DNA that can be modeled with $n = 87$, [19], [23]. Well-tested energy parameters are available, and specific local minima for dC are known experimentally. The energy function $F(\mathbf{x})$ is typically written in terms of the *internal* variables (interatomic distances r_{ij}, bond lengths b_i, bond angles θ_i, and dihedral angles τ_i), which are in turn computed from $\mathbf{x}$ through geometric relations [19]. It has the form:

$$(30)\quad \begin{aligned} F = &\sum_{i,j\in S_{NB}} \left[-\frac{A_{ij}}{r_{ij}^6} + \frac{B_{ij}}{r_{ij}^{12}} + \frac{Q_iQ_j}{r_{ij}} \right] + \sum_{i\in S_B} S_i(b_i - \bar{b}_i)^2 \\ &+ \sum_{i\in S_{BA}} K_i(\cos\theta_i - \cos\bar{\theta}_i) + \sum_{i\in S_{DA}}{}' \sum_m V_{m_i}(1 + \cos m\tau_i). \end{aligned}$$

[2] The costs for the Hessian/vector products $H\mathbf{d}$ (see step 3 of Algorithm TN2) and solutions of $M\mathbf{z} = \mathbf{r}$ (see (22) and (28)) depend on the problem structure. The finite-difference approximation to $H\mathbf{d}$ (see footnote 1) will generally provide a time advantage, without sacrifice in performance, if the "analytic" $H\mathbf{d}$ calculation is considerably more costly than one gradient evaluation. The $M\mathbf{z} = \mathbf{r}$ solution depends on the sparsity pattern of M and can be as low as $O(n)$ [21].

The first component represents the van der Waals (attraction/repulsion) and Coulombic interactions for atom pairs (i,j), $i<j$, in the nonbonded set S_{NB}. The second and third energy terms penalize deviations from reference bond lengths $(\bar{b}_i)$ or bond angles $(\bar{\theta}_i)$ for the bonds and angles in the sets S_B and S_{BA}, respectively. The last term accounts for rotations of two groups of atoms about the bond connecting them and associates with each such dihedral angle in S_{DA} a periodicity m (typically $m = 2,3,6$) and barrier height V_m. The symbols A, B, Q, S, and K are energy parameters associated with a particular pair interaction, bond, or angle type (e.g., O$-$O where O = oxygen). M_{PE} is assembled from the second derivatives of the bond length, bond angle, and dihedral angle terms.

Many minimization starting conformations were investigated for dC, and the three different choices (**X1**, **X2**, **X3**), examined in Tables 1–4, are representative of different energies and eigenvalue distributions (see Tables 3 and 4). All Hessians and preconditioners are indefinite at the starting points. Tables 1 and 2 show minimization results from the three points: in Table 1 with the truncation parameter $\delta = 1$ (see (21)), and in Table 2 with $\delta = 10^{-8}$. The total number of Newton iterations, PCG iterations, and function evaluations required for convergence is given. The minimum, maximum, and median values for $\|E\|_\infty$, computed over all Newton iterations, and the number of Newton iterations for which $\|E\|_\infty > 0$ ("modified iterations") provide the MC information. Each run from the **X1**, **X2**, and **X3** groups produced the minima **X6**, **X5**, and **X7**, respectively. Tables 3 and 4 provide eigenvalue information for H and M at these six points.

TABLE 1
Molecular model 1: *Deoxycytidine* (dC), $n = 87$ ($\delta = 1.0$ *in truncation criterion*).

	Run	Newton Itns.	PCG Itns.	F & G Evals.	$\|E\|_\infty$ Min.	$\|E\|_\infty$ Max.	$\|E\|_\infty$ Med.	Modified Itns.
	GMW	15	71	38	1.3E+01	1.3E+01	1.3E+01	1
(**X1**)	SE(a)	17	84	43	2.4E+00	5.6E+00	4.0E+00	2
	SE(b)	16	81	41	5.6E+00	5.6E+00	5.6E+00	1
	GMW	19	79	48	1.4E+01	1.4E+01	1.4E+01	1
(**X2**)	SE(a)	21	79	50	7.2E+00	7.2E+00	7.2E+00	1
	SE(b)	19	86	46	7.2E+00	7.2E+00	7.2E+00	1
	GMW	13	48	28	1.3E+01	1.3E+01	1.3E+01	1
(**X3**)	SE(a)	13	49	31	6.1E+00	7.2E+00	6.7E+00	2
	SE(b)	14	57	27	7.2E+00	7.2E+00	7.2E+00	1

From Table 1 ($\delta = 1$), we note that overall convergence to a local minimum is very rapid: $< n/4$ outer iterations, $< n$ inner iterations, and $\sim n/2$ function evaluations. M_{PE} is modified in only one or two Newton iterations (typically the first and second), and in those cases $e_j > 0$ only for the last three or four factorization steps ($j \geq 83$). The values of $\|E\|_\infty$ are larger for GMW than SE, by about a factor of two, but in both strategies $\|E\|_\infty$ is greater than $\lambda_{\min}$ by one or two orders of magnitude. When a modification occurs in only one Newton iteration, $\|E\|_\infty$ corresponds to $\lambda_{\min}$ of M at the starting point (Table 3). When two such modifications are made, the recorded value for $\|E\|_\infty$ corresponds to a $\lambda_{\min}$ of M (at another Newton iteration) that is slightly smaller in magnitude than $\lambda_{\min}$ at the starting point. (In the second iteration of SE(a) for **X1**, $\lambda_{\min} = -0.22$ versus $\|E\|_\infty = 2.4$; in the second iteration of SE(a) for **X3**, $\lambda_{\min} = -0.01$ versus $\|E\|_\infty = 6.1$.) Overall, the smaller SE modifications do not produce a systematic reduction of PCG iterations for the modified iterations.

TABLE 2
Molecular model 1: *Deoxycytidine* (dC), $n = 87$ ($\delta = 10^{-8}$ *in truncation criterion*).

	Run	Newton Itns.	PCG Itns.	F & G Evals.	$\|E\|_\infty$ Min.	$\|E\|_\infty$ Max.	$\|E\|_\infty$ Med.	Modified Itns.
(X1)	GMW	12	207	29	1.3E+01	1.3E+01	1.3E+01	1
	SE(a)	13	230	31	5.6E+00	5.6E+00	5.6E+00	1
	SE(b)	12	191	29	5.6E+00	5.6E+00	5.6E+00	1
(X2)	GMW	20	299	58	1.4E+01	6.6E+01	4.0E+01	2
	SE(a)	18	278	45	7.2E+00	7.2E+00	7.2E+00	1
	SE(b)	18	287	36	7.2E+00	7.2E+00	7.2E+00	1
(X3)	GMW	13	189	34	1.3E+01	1.3E+01	1.3E+01	1
	SE(a)	12	156	23	4.7E+00	7.2E+00	6.0E+00	2
	SE(b)	13	177	31	7.2E+00	7.2E+00	7.2E+00	1

TABLE 3
Eigenvalue information for dC *starting points* ($F(\mathbf{X1}) = 1.4$; $F(\mathbf{X2}) = 1.8 \times 10^3$; $F(\mathbf{X3}) = 5.1 \times 10^4$).

Point		$\lambda_{\min}$	$\lambda_{\max}$	#NEG
(X1)	H	−1.4E+00	2.9E+03	2
	M	−3.2E−01	2.8E+03	1
(X2)	H	−1.2E+04	1.8E+05	7
	M	−1.2E−01	2.9E+03	1
(X3)	H	−8.2E+05	1.1E+07	9
	M	−1.2E−01	2.8E+03	1

TABLE 4
Eigenvalue information for dC *minima* ($F(\mathbf{X5}) = -6.8$; $F(\mathbf{X6}) = -5.7$; $F(\mathbf{X7}) = -5.0$).

Point		$\lambda_{\min}$	$\lambda_{\max}$	#NEG
(X5)	H	7.7E−02	3.0E+03	0
	M	3.7E−01	3.0E+03	0
(X6)	H	3.8E−02	2.9E+03	0
	M	2.6E−01	2.9E+03	0
(X7)	H	2.1E−02	3.1E+03	0
	M	3.8E−01	3.1E+03	0

Only about one to three PCG iterations per Newton iteration are performed for all Newton iterations. The differences between the two SE versions are small.

When the truncation parameter is made more strict ($\delta = 10^{-8}$), Table 2 shows a large increase in the number of required PCG iterations (a factor of three or more). This demonstrates the effectiveness of the truncated Newton approach in allowing larger residuals in many cases. The number of function and gradient evaluations is usually decreased as well.

Overall, both the GMW and SE strategies perform quite similarly in this example despite the fact that $\|E\|_\infty$ is systematically greater for GMW than for SE. This suggests that the size of E alone is not a simple factor correlated with better performance in TN; larger modifications to M may subsequently produce equally effective search directions. Indeed, in earlier tests, we encountered a case where $\|E\|_\infty$ of GMW was larger by nearly two orders of magnitude than that of SE; yet minimization performance of the GMW version was more rapid overall. Although the differences here are not large, the GMW version appears slightly more efficient in the usual TN context of $\delta = 1$. Between the two SE versions, performance differences are small. Analysis shows that allowing e_{j-1} to exceed e_j can slightly increase the average modification per Newton step but may cause fewer elements to be modified. Systematic effects of this difference cannot be noted here since few nonzero diagonal modifications are involved, all of which occur in the last few Cholesky factorization steps.

The rapid minimization progress and its insensitivity to the MC strategy may be attributed to the fact that this optimization problem is straightforward, with distinct convex conformational regions. In almost all runs, H becomes positive definite after the first Newton iteration and remains positive definite. M_{PE} is an excellent preconditioner (nonpreconditioned runs required a factor of ten or more PCG iterations [22]) and is positive definite except during one or two Newton iterations. As later results suggest, this overall performance may be related to the existence of only one *isolated* negative eigenvalue for M_{PE} near the starting point; all three MC strategies modify only few diagonals and are equally successful in our TN context.

4.3. Water clusters. Our second model of water clusters [20] has an energy function similar to (30) except that the van der Waals term is summed over O–O atom pairs only, the coulombic interactions are summed over all intermolecular pairs (O–O, O–H, and H–H), and there is no dihedral angle term (H = hydrogen). The physical picture is entirely different from that of dC, since the interaction among the different water molecules is long range and the many local minima (close in energy) correspond to different hydrogen-bonding networks. The complexity of these networks increases as the cluster size grows. This property produces a Hessian for the water model that is often indefinite. Furthermore, the same definition of local M_{PE} does not contain the major energy contributions. The electrostatic potential that leads to formation of hydrogen bonds is dominant here. However, it is not practical to formulate a preconditioner from these long-range interactions, especially in the context of large-scale truncated Newton minimization. In practice, these features are reflected in TN minimization by frequent termination of the inner loop when a direction of negative curvature is detected and by nonzero MC modifications at every Newton iteration. In combination with the difficulty of choosing good starting points (i.e., predicting the hydrogen bonding patterns a priori), this minimization problem is far more difficult than dC and requires a large number of iterations and function calls.

In all water runs, the truncation parameter δ is set to unity and the limiting number of inner iterations (*per* outer iteration) is set to n to make the MC comparisons as fair as possible. Results for a water dimer ($n = 18$) are summarized in Table 5. Run (A) uses M_{PE}. Run (B), chosen for diagnostic purposes, adds to M_{PE} nonlocal O–O interactions. This adds an off-diagonal 3×3 block to the block-diagonal M_{PE} composed of two 9×9 blocks. There is only one minimum for the dimer corresponding to a linear hydrogen-bonded pair [20].

Run (A) shows convergence in about 40 Newton iterations for all MC versions. GMW requires a smaller number of inner iterations and function evaluations ($\sim$ 80 percent) in relation to the SE runs. The size of $\|E\|_\infty$ for GMW is on the order of the prescribed parameter τ of (13), since very small trial diagonals—$O(10^{-13} - 10^{-20})$—appear toward the end of the factorization ($j > 15$). The SE versions switch to Phase 2 early, producing more diagonal modifications but also avoiding formation of very small tentative diagonals. Table 6(a) provides an illustration of this behavior.

In contrast, run (B) shows that *overall* performance of all three MC strategies is very similar. The mean GMW $\|E\|_\infty$ is now *greater* by about one order of magnitude

TABLE 5
Molecular model 2: water dimer, $n = 18$.

	Run	Newton Itns.	PCG Itns.	F & G Evals.	$\|E\|_\infty$ Min.	$\|E\|_\infty$ Max.	$\|E\|_\infty$ Med.	Modified Itns.
(A)	GMW	38	244	100	1.5E−05	1.6E−05	1.6E−05	38
	SE(a)	47	322	132	9.5E−01	1.1E+03	5.5E+02	47
	SE(b)	43	296	120	9.5E−01	1.1E+03	5.5E+02	43
(B)	GMW	46	294	129	1.0E+01	6.7E+03	3.4E+03	46
	SE(a)	45	297	127	6.0E+00	9.2E+02	4.6E+02	45
	SE(b)	45	297	134	1.5E+01	9.2E+02	4.7E+02	45

than SE. Thus, the noted tendency of GMW to begin diagonal modifications at later factorization steps produces here an overall larger $\|E\|_\infty$. Table 6(b) illustrates this behavior in comparison to run (A).

A minimization series of water clusters in increasing sizes was then performed. These experiments correspond to case (A) of the water dimer, using M_{PE}. Starting configurations are constructed pseudorandomly in the computational domain [20], and are thus high in energy and far from optimal structures. The various geometric possibilities that different cluster sizes can adopt [20], in combination with the random aspect of choosing the starting points, produce differences in minimization performance that are not always correlated to the cluster size.

Table 7 shows performance for water clusters of dimensions $n = 27$ to $n = 576$, and Table 8 provides eigenvalue information for the corresponding starting and final points (including the water dimer). In the reported runs for each dimension, the resulting minima are very close in energy although they may not be identical. In such cases, corresponding eigenvalue information at the final points is very similar for each group, and the reported values in Table 8 are representative.

In comparison with results for the water dimer, similar behavior in the size of $\|E\|_\infty$ can be observed for larger water models, but a new dimension-dependent trend emerges among the MC versions. While the GMW version appears more efficient for smaller dimensions in the number of iterations and function evaluations, the SE strategy, in particular SE(b), shows better performance with increasing size. Differences in the number of PCG iterations, in particular, become large (by factors of two to more than four) in many cases.

Analysis reveals the following explanation. The Hessian is indefinite at the starting point and remains so for many iterations; thus, the inner loop often ends with detection of negative curvature. M_{PE} is also indefinite, with approximately $n/9$ negative eigenvalues (all very close to zero) throughout the run, forcing nonzero modifications at every Newton iteration. Now, GMW produces $\|E\|_\infty$ sizes on the order of the tolerance τ ($O(10^{-5})$), while the Gerschgorin estimates of SE lead to much larger modifications ($O(10^3)$), similar to the situation illustrated in Table 6(a). The resulting GMW search directions often produce very slow progress for many iterations, and the number of H's negative eigenvalues decreases slowly; it is likely that these directions are close to perpendicular to the gradient. When H becomes positive definite, the truncation criterion is stricter (see (21)) and many PCG iterations are then involved. The SE strategy enters positive definite regions of H more quickly and typically displays more rapid progress thereafter. Version SE(b) performs better than SE(a) because diagonal modifications are made less often and a smaller number of PCG iterations generally results.

TABLE 6 (a)
MC *analysis for the first Newton iteration of Table* 5, *run* (A).

j	GMW d_j	GMW d_j+e_j	SE(a) d_j	SE(a) d_j+e_j	SE(b) d_j	SE(b) d_j+e_j
10			350	1452	350	1452
11			1102	2105	1102	2005
12	−2.1E−13	1.5E−05	266	1368	266	837
13	3.6E−15	1.5E−05	86	1189	77	230
14	1.6E+02	0	349	1451	288	288
15	−5.0E−14	1.5E−05	71	1173	31	62
16	3.6E−15	1.5E−05	86	1188	72	201
17	−3.6E−15	1.5E−05	341	1444	209	209
18	−5.0E−14	1.5E−05	70	1172	13	13
	$\|\|E\|\|_\infty = 1.5E-05$		$\|\|E\|\|_\infty = 1103$		$\|\|E\|\|_\infty = 1103$	

TABLE 6 (b)
MC *analysis for the tenth Newton iteration of Table* 5, *run* (B).

j	GMW d_j	GMW d_j+e_j	SE(a) d_j	SE(a) d_j+e_j	SE(b) d_j	SE(b) d_j+e_j
14	*		83	128	94	169
15	− 0.2	0.2	0.3	46	3	20
16	*		12	58	12	15
17	− 13	13	22	68	*	
18	−104	104	−0.5	45	−0.6	0.02
	$\|\|E\|\|_\infty = 208$		$\|\|E\|\|_\infty = 46$		$\|\|E\|\|_\infty = 75$	

* Entries leading to zero modifications were not recorded.

Performance with a stricter truncation criterion was very similar since only the last several iterations (that do not end in detection of negative curvature) are affected, but for these the allowed residual is already very small. Performance with larger values of τ (the minimum allowable value of each positive diagonal modification), such as by two to five orders of magnitude, produced somewhat better overall progress for the GMW runs, but progress is still not competitive with SE. Performance with *smaller* τ led to some underflow/overflow for the larger dimensions. This suggests that here ill-conditioned preconditioners $\overline{M}$ are disadvantageous. For the structure in these examples, larger tolerances for GMW are an effective way to increase $||E||_\infty$, whereas in SE, $||E||_\infty$ is larger—even with small τ—due to the use of the Gerschgorin estimates.

Why does the GMW version display faster convergence for smaller dimensions? The number of negative eigenvalues and percentage of iterations when H is indefinite seem to play key roles in steering the minimization progress. Table 8 shows that H has about $n/3$ negative eigenvalues at the starting point, with a $\lambda_{\min}$ of magnitude $O(10^0-10^1)$ and a $\lambda_{\max}$ of $O(10^3)$. The number of M's negative eigenvalues is about $n/9$, but they are all very small in magnitude and clustered. Thus, when only a few tentative diagonals are small (say, one to three), GMW is more efficient in terms of iterations and function evaluations because its small modifications apparently do not affect the "quality" of the computed search direction for TN dramatically; as the number of small tentative diagonals increases and more resulting diagonals have magnitude τ, 10^{-5}, we observe slower progress. The SE strategy detects indefiniteness early in the factorization (Phase 1 is typically false after about ten steps) and modifies the diagonals more substantially. This appears advantageous as the size and complexity of the problem increase due to the better conditioning of the preconditioner. In other

TABLE 7
Molecular model 2: Water clusters.

n	Run	Newton Itns.	PCG Itns.	F & G Evals.	$\|E\|_\infty$ Min.	Max.	Med.	Modified Itns.
27	GMW	38	506	67	1.5E−05	6.6E−01	3.3E−01	38
	SE(a)	63	1147	94	3.3E+02	1.6E+03	1.0E+03	63
	SE(b)	68	1053	106	5.0E+02	1.5E+03	1.0E+03	68
36	GMW	70	1400	114	1.5E−05	1.8E−05	1.7E−05	70
	SE(a)	71	1783	110	3.9E+02	1.5E+03	9.5E+02	71
	SE(b)	75	1554	128	3.9E+02	1.6E+03	1.0E+03	75
45	GMW	81	2256	148	1.5E−05	1.9E−05	1.7E−05	81
	SE(a)	99	3606	159	5.4E+02	1.6E+03	1.1E+03	99
	SE(b)	94	2598	162	9.6E+02	1.5E+03	1.2E+03	94
54	GMW	116	4744	212	1.5E−05	1.9E−05	1.7E−05	116
	SE(a)	116	5554	179	9.9E+02	1.6E+03	1.3E+03	116
	SE(b)	108	3886	158	1.1E+03	1.6E+03	1.4E+03	108
72	GMW	85	4771	157	1.5E−05	1.9E−05	1.7E−05	85
	SE(a)	64	3155	102	8.0E+02	1.6E+03	1.2E+03	64
	SE(b)	54	1848	89	8.4E+02	1.6E+03	1.2E+03	54
90	GMW	112	8524	200	1.5E−05	1.9E−05	1.7E−05	112
	SE(a)	85	5804	133	1.0E+03	1.6E+03	1.3E+03	85
	SE(b)	85	4879	154	1.1E+03	1.6E+03	1.3E+03	85
108	GMW	143	12941	270	1.5E−05	1.3E+01	6.4E+00	143
	SE(a)	101	7573	162	1.2E+03	1.5E+03	1.4E+03	101
	SE(b)	97	6285	174	1.0E+03	1.7E+03	1.4E+03	97
144	GMW	132	15702	249	1.6E−05	2.3E−05	1.9E−05	132
	SE(a)	118	12217	217	1.2E+03	1.6E+03	1.4E+03	118
	SE(b)	97	8631	163	1.2E+03	1.6E+03	1.4E+03	97
180	GMW	88	10314	236	1.5E−05	1.8E−05	1.6E−05	88
	SE(a)	47	3816	104	1.3E+03	1.6E+03	1.5E+03	47
	SE(b)	47	2657	107	1.1E+03	1.7E+03	1.4E+03	47
243	GMW	71	8173	212	1.4E−05	1.7E−05	1.6E−05	71
	SE(a)	56	5828	104	1.1E+03	1.6E+03	1.4E+03	56
	SE(b)	60	4880	137	1.1E+03	1.8E+03	1.5E+03	60
324	GMW	104	20543	277	1.4E−05	1.9E−05	1.7E−05	104
	SE(a)	53	4450	120	1.2E+03	1.7E+03	1.5E+03	53
	SE(b)	55	4581	123	1.1E+03	1.7E+03	1.4E+03	55
576	GMW	100	23502	407	1.4E−05	1.6E−05	1.5E−05	100
	SE(a)	66	9040	195	1.3E+03	1.6E+03	1.5E+03	66
	SE(b)	57	7732	139	1.4E+03	1.7E+03	1.6E+03	57

words, when H is mainly positive definite, GMW does better because M is closer to H, whereas when H is often indefinite SE does better because M is better conditioned.

4.4. The trigonometric function. A well-known test function for optimization, known as "trigonometric," offers a problem structure with many negative eigenvalues of larger magnitudes. This function of variable dimension n is given as [12]

$$F(\mathbf{x}) = \sum_{j=1,\ldots,n} \left(n - \sum_{i=1}^{n} \cos x_i + j(1 - \cos x_j) - \sin x_j \right)^2 . \tag{31}$$

At $\mathbf{x}_0 = (1/n, \ldots, 1/n)^T$, H is indefinite. We study four sets of minimization runs from $\mathbf{x}_0$, for $n = 50$, 100, 250, and 1000. For each set, we use the diagonal, tridiagonal, and pentadiagonal elements of the Hessian to construct preconditioners (denoted as D, T, and P). The truncation parameter δ is set to unity, and the limiting number

TABLE 8
Eigenvalue information for water clusters.

n		$\mathbf{x}_0$ $\lambda_{\min}$	$\lambda_{\max}$	# NEG	$\mathbf{x}_*$ $\lambda_{\min}$	$\lambda_{\max}$	# NEG
18	H	−2.4E+00	2.2E+03	3	9.8E−02	2.2E+03	0
	M	−1.6E−13	2.1E+03	1	−1.7E−14	2.2E+03	2
27	H	−4.5E+01	2.3E+03	8	5.4E−02	2.3E+03	0
	M	−1.3E−13	2.3E+03	2	−1.2E−14	2.2E+03	2
36	H	−4.5E+01	2.3E+03	11	1.5E−02	2.3E+03	0
	M	−1.5E−13	2.3E+03	3	−2.8E−14	2.2E+03	5
45	H	−4.5E+01	2.3E+03	14	1.9E−04	2.3E+03	0
	M	−1.5E−13	2.3E+03	4	−2.8E−14	2.2E+03	4
54	H	−5.6E+01	2.3E+03	17	9.6E−03	2.3E+03	0
	M	−1.5E−13	2.3E+03	7	−2.1E−14	2.2E+03	5
72	H	−5.6E+01	2.3E+03	23	8.9E−03	2.3E+03	0
	M	−1.5E−13	2.3E+03	9	−3.6E−14	2.2E+03	6
90	H	−1.1E+02	2.3E+03	28	1.2E−03	2.3E+03	0
	M	−1.5E−13	2.3E+03	12	−2.8E−14	2.2E+03	7
108	H	−7.5E+01	2.3E+03	34	4.1E−03	2.3E+03	0
	M	−1.6E−13	2.3E+03	12	−2.8E−14	2.2E+03	9
144	H	−9.6E+01	2.3E+03	45	8.3E−03	2.3E+03	0
	M	−1.5E−13	2.3E+03	17	−3.4E−14	2.2E+03	14
180	H	−1.1E+01	2.4E+03	54	3.3E−03	2.3E+03	0
	M	−1.7E−13	2.3E+03	24	−3.3E−14	2.2E+03	16
243	H	−1.1E+02	2.4E+03	70	2.6E−03	2.3E+03	0
	M	−2.5E−13	2.3E+03	31	−2.9E−14	2.2E+03	25

of PCG iterations (per Newton step) is set to ten.[3] Typically, the residual is very small after a small number of PCG iterations, so this PCG limit has only a small effect. Since the residual is reduced rapidly in the inner loop, a smaller δ value also has little effect on performance. Since very similar linear systems are solved at every step, we expect our MC comparisons to reflect systematic differences at each Newton step. For all runs, we use a stricter overall minimization termination criterion of $\|\mathbf{g}\|/\sqrt{n} \leq 10^{-12} * \max\{1, \|\mathbf{x}\|\}$; since $\|\mathbf{g}(\mathbf{x}_0)\|$ is already small, we wanted to ensure continuation of the minimization until quadratic convergence of $\|\mathbf{g}\|$ is observed.

Table 9 shows results for the four sets of runs. Table 10 lists $\lambda_{\min}$ and $\lambda_{\max}$ for H and for P at both $\mathbf{x}_0$ and $\mathbf{x}_*$ (the solution). The corresponding eigenvalues for D and T are very similar. (This is probably a consequence of the small magnitudes of off-diagonals with respect to diagonal elements: typical values of γ and ξ are $\gamma \sim 2.0$ and ξ of $O(10^{-2} - 10^{-1})$.) We note that both H and M have 60 percent of their eigenvalues negative at $\mathbf{x}_0$. A complete spectral analysis shows that all eigenvalues have magnitudes $O(10^{-1} - 10^{-2})$, most of them of $O(10^{-1})$. For example, for $n = 100$, the 61 negative eigenvalues of P range between -0.59 and -0.02. All eigenvalues at $\mathbf{x}_0$ for H and M are concentrated in the interval $[-1.0, 1.5]$. At $\mathbf{x}_*$, H and M are positive definite, with $\lambda_{\min}$ of $O(10^{-2})$ and $\lambda_{\max}$ approximately 3.0 for H and 2.0 for M.

The runs with preconditioner D show that performance with the GMW strategy is more rapid for each dimension. The number of "modified" iterations in GMW is smaller for $n > 50$ and equal to SE for $n = 50$. The size of $\|E\|_\infty$ is typically greater for GMW by a factor of two. This relation is expected since $\theta = 0$ and $|\tilde{d}_j| > \tau$ (see (13) and (17)). Thus, GMW generally modifies tentative negative diagonals m_{jj} to

[3] This limit is imposed to mimic a practical situation where H is dense and the inner iterations are costly when $H\mathbf{d}$ is computed analytically.

TABLE 9
Trigonometric function minimization.

			Newton	PCG	F & G		$\|E\|_\infty$		Modified
n	M	Run	Itns.	Itns.	Evals.	Min.	Max.	Med.	Itns.
		GMW	15	79	22	1.1E+00	1.1E+00	1.1E+00	1
50	(D)	SE(a)	15	88	41	5.5E−01	5.5E−01	5.5E−01	1
		SE(b)	22	84	38	2.1E−03	5.5E−01	2.8E−01	9
		GMW	22	102	54	3.0E−02	1.5E+00	7.4E−01	14
50	(T)	SE(a)	15	108	20	4.2E−02	6.0E−01	3.2E−01	7
		SE(b)	13	107	18	3.7E−02	6.0E−01	3.2E−01	6
		GMW	30	125	52	8.0E−02	6.1E+00	3.1E+00	22
50	(P)	SE(a)	14	111	38	9.0E−02	6.2E−01	3.5E−01	6
		SE(b)	13	101	17	5.7E−02	6.2E−01	3.4E−01	8
		GMW	19	65	36	1.3E−01	1.1E+00	6.3E−01	13
100	(D)	SE(a)	47	109	83	3.0E−03	5.7E−01	2.8E−01	34
		SE(b)	30	79	41	6.9E−03	5.7E−01	2.9E−01	15
		GMW	28	114	54	6.3E−02	1.7E+00	8.7E−01	19
100	(T)	SE(a)	15	98	22	2.0E−02	5.9E−01	3.1E−01	8
		SE(b)	15	107	36	2.8E−02	5.9E−01	3.1E−01	7
		GMW	58	139	92	8.8E−02	7.7E+00	3.9E+00	51
100	(P)	SE(a)	14	100	93	4.4E−02	6.0E−01	3.2E−01	7
		SE(b)	13	100	21	2.4E−02	6.0E−01	3.1E−01	6
		GMW	35	106	91	1.1E−01	1.2E+00	5.8E−01	28
250	(D)	SE(a)	47	134	71	4.3E−02	5.8E−01	2.9E−01	33
		SE(b)	126	197	183	3.2E−05	5.8E−01	2.9E−01	115
		GMW	45	138	88	8.4E−03	1.4E+00	7.2E−01	36
250	(T)	SE(a)	20	109	54	7.1E−02	5.9E−01	3.0E−01	10
		SE(b)	61	136	100	6.1E−03	5.9E−01	3.0E−01	52
		GMW	50	166	106	1.5E−02	7.8E+00	3.9E+00	40
250	(P)	SE(a)	17	108	40	2.0E−02	5.9E−01	3.1E−01	8
		SE(b)	23	124	56	2.0E−02	5.9E−01	3.1E−01	11
		GMW	55	151	128	2.2E−03	1.7E+00	8.3E−01	49
1000	(D)	SE(a)	150	220	229	3.1E−05	5.8E−01	2.9E−01	137
		SE(b)	223	276	319	2.9E−03	5.8E−01	2.9E−01	215
		GMW	86	206	181	2.6E−01	1.8E+00	1.0E+00	79
1000	(T)	SE(a)	34	95	74	4.5E−03	5.8E−01	2.9E−01	26
		SE(b)	145	184	205	4.5E−03	5.8E−01	2.9E−01	140
		GMW	50	133	126	6.7E−02	1.9E+00	9.9E−01	46
1000	(P)	SE(a)	21	94	60	5.9E−03	5.9E−01	3.0E−01	12
		SE(b)	41	109	61	5.9E−03	5.9E−01	3.0E−01	33

The symbols (D), (T), and (P) denote diagonal, tridiagonal, and pentadiagonal preconditioners, respectively.

TABLE 10
Eigenvalue information for the trigonometric function.

		$\mathbf{x}_0$			$\mathbf{x}_*$		
n	$\lambda_{\min}$	$\lambda_{\max}$	#NEG	$\lambda_{\min}$	$\lambda_{\max}$	#NEG	
50	H	−5.7E−01	1.4E+00	30	5.3E−02	2.7E+00	0
	M	−6.0E−01	1.4E+00	31	6.4E−02	2.2E+00	0
100	H	−5.8E−01	1.5E+00	60	6.3E−02	2.8E+00	0
	M	−5.9E−01	1.4E+00	61	6.5E−02	2.2E+00	0
250	H	−5.8E−01	1.5E+00	152	4.5E−02	2.8E+00	0
	M	−5.9E−01	1.5E+00	152	6.3E−02	2.0E+00	0
1000	H	−5.8E−01	1.5E+00	607	4.8E−02	2.7E+00	0
	M	−5.8E−01	1.5E+00	608	7.6E−02	2.0E+00	0

$|m_{jj}|$, and SE produces elements d_j of magnitude τ. The small diagonals appear to produce search directions that lead to slower minimization progress (more below).

As off-diagonal elements are added to M, more iterations, function evaluations, and nonzero modifications can be observed for GMW in comparison to SE. The ratio of the largest elements in E in GMW versus SE is in the range 2–3 for T and 3–13 for P. Version SE(a) produces the most rapid performance as the size of the problem increases. Figure 2 illustrates the difference in minimization progress by comparing $\lambda_{\min}$ with $\|E\|_\infty$ for the runs with T and $n = 250$. During the first five Newton iterations, the number of negative eigenvalues of M in the GMW version is reduced slowly from 152: $\{152, 135, 102, 44, 55, 31\}$. In comparison, the SE(a) and SE(b) sequences of negative eigenvalues per Newton iteration are $\{152, 5, 2, 0, 0, 0\}$ and $\{152, 5, 4, 0, 0, 0\}$, respectively. At later iterations, the number of negative eigenvalues in GMW ranges from zero to 65, while the range for the SE versions is between zero and eight.

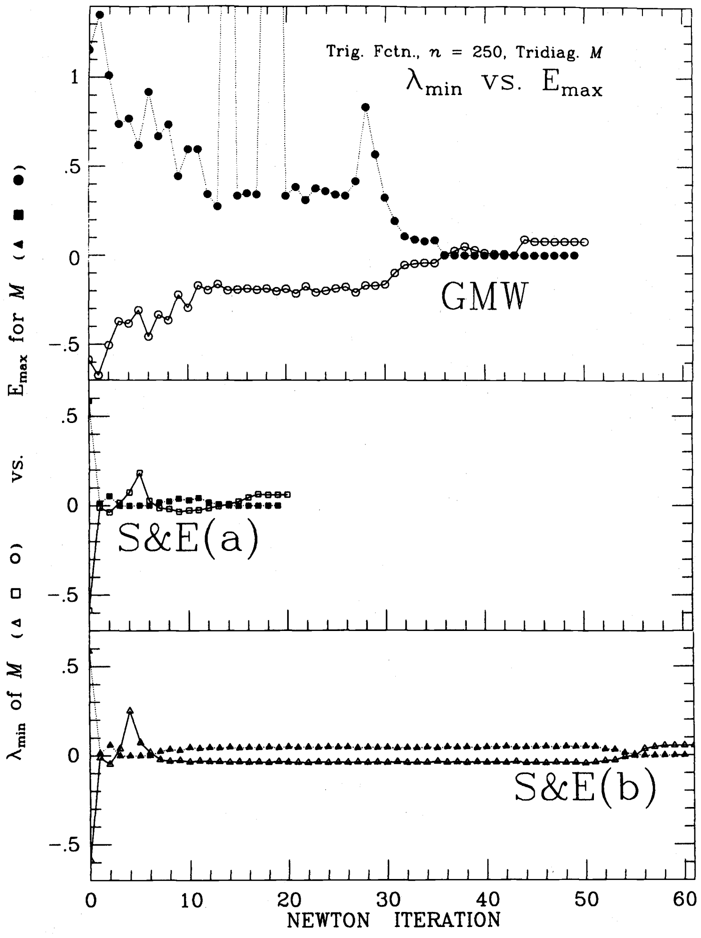


FIG. 2. *$\lambda_{\min}$ versus $\|E\|_\infty$ for the trigonometric function.*

For the first Newton iteration, the modification schedule is quite similar for all MC versions: GMW modifies diagonals $99 \le j \le 250$, and the SE runs enter Phase 2 at $j = 97$. Two PCG iterations are performed for GMW, and one PCG iteration is performed for SE(a) and SE(b), until a direction of negative curvature is detected. Notable differences emerge thereafter. At the second Newton iteration, GMW modifies negative tentative diagonals $96 \leq j \leq 250$ of magnitude range $O(10^{-1} - 10^{-3})$. Both SE versions enter Phase 2 at $j = 153$ and modify a few small negative tentative diagonals as well as small positive diagonal candidates of $O(10^{-2} - 10^{-4})$; subsequent diagonals do not become negative. This pattern for all MC versions repeats for several Newton iterations, involving fewer and fewer diagonal modifications at each iteration.

The tendency of small diagonal entries to form during the MC factorization is reversed for the GMW and SE versions in the case of diagonal and nondiagonal preconditioners. GMW handles this by specifying larger modifications in the diagonal case, while SE applies a different two-phase modification schedule in the nondiagonal cases. As observed in the water minimization, a systematic trend of small, positive diagonals of $O(\tau)$ slows minimization progress in TN. The clustered eigenvalue structure of the trigonometric function appears to work more favorably with the SE MC scheme. Interestingly, it has been observed that SE produces the most significant improvement in $\|E\|_\infty$ over GMW for problems where the eigenvalues are clustered between -1 and 1 [25].

5. Summary.

The GMW and SE MC factorizations were examined in the context of modifying indefinite preconditioners in a truncated Newton algorithm. No pivoting versions were examined since a variable reordering is determined at the beginning of the optimization procedure to minimize fill-in for the sparse preconditioner. The two factorizations differ both in their formulas for diagonal modifications and in the manner by which these formulas are activated. While GMW subjects all tentative diagonals to the same modification formula, SE employs a two-phase strategy. Phase 1 denotes the state when all diagonal modifications, e_j, are zero (positive definite matrices always remain in Phase 1), and Phase 2 is entered when a phase test indicates that a diagonal element will become negative or very "small." Only at Phase 2 is the Gerschgorin-based formula of SE applied. In the last two steps, explicit eigenvalue information is used. Whereas the phase test in SE requires that the diagonal elements of the matrix be updated at every step, this is not a requirement in GMW. While GMW requires the full matrix a priori, SE only needs the diagonal at the start. A variation of the SE factorization was suggested here to produce the same modification formula for all diagonal elements. In addition, an SE variation in which the modification e_j was not required to be as large as e_{j-1} was examined.

Our three test problems, with variations of size, preconditioners, starting points, and truncation parameters, involve three different eigenvalue distributions that are typically encountered in practice: an isolated negative eigenvalue, several near-zero positive and negative eigenvalues, and a large percentage of negative eigenvalues. In the first case, represented by a molecular model with distinct and well-separated energy minima, nonzero diagonal modifications to M were required in only one or two Newton iterations. The GMW and SE factorizations behaved very similarly, though GMW was slightly more efficient despite the typically larger $\|E\|_\infty$ (factor of two). In the second case, a molecular ensemble possessing numerous and clustered minima with a characteristically indefinite potential surface, nonzero modifications were made in all Newton iterations. GMW appeared more efficient for smaller dimensions but SE(b) (the version where e_j was not required to exceed e_{j-1}) became much more efficient as size increased; $\|E\|_\infty$ was typically around the set tolerance τ, $O(10^{-5})$ for GMW, and $O(10^2 - 10^3)$ for SE. In the third case of the trigonometric function, nonzero modifications were involved in some Newton iterations. GMW demonstrated more rapid performance for diagonal preconditioners of various dimensions, but SE(a) (the version where $e_j \geq e_{j-1}$) produced better performance as off-diagonals were added to the preconditioner. $\|E\|_\infty$ was typically larger for GMW by a factor of two to ten.

Our results suggest that differences in performance between the GMW and SE MC factorizations depend on such problem characteristics. The following conclusions may be drawn from the test problems examined *and* our special preconditioner-modifier context.

(1) The size of $\|E\|_\infty$ may not be a crucial factor governing minimization performance. Although in some cases $\|E\|_\infty$ from SE is smaller than that of GMW, minimization progress is not directly correlated with this number, especially for problems with few isolated negative eigenvalues. This behavior is both a consequence of the indirect effect of $\|E\|_\infty$ on the computed search direction and the inherent approximation in the truncated Newton approach.

(2) The trends in the size of $\|E\|_\infty$ for the GMW and the SE schemes may be distinct for the dense and sparse matrix cases. Whereas dense test problems reveal that the SE MC often produces smaller values of $\|E\|_\infty$ than GMW [25], sparse matrices may exhibit an opposite trend. The Gerschgorin-based estimates of SE may lead to much larger modifications than GMW in sparse systems, especially in problems with many near-zero eigenvalues. However, the possibility that these observations stem from the eigenvalue patterns of the matrices—whether the matrices are sparse or dense—cannot be excluded.

(3) Producing very small $\|E\|_\infty$ (i.e., so that $M + E$ is nearly singular) is usually disadvantageous in our context, unless M has only a few negative eigenvalues. Such near-singular "effective preconditioners" often require many inner (PCG) iterations at each Newton step and produce very slow progress overall. This slow progress is most pronounced for large dimensions. This observation suggests that larger tolerances for the minimum-allowed $\|E\|_\infty$ would be more effective in these cases.

(4) The largest difference between the two factorizations appears to be related to their different modification *schedules*. In many cases, the GMW diagonal modifications begin at later stages of the factorization than SE; tentative diagonal values may then grow more negative and require larger modifications later on. With modifications at earlier stages of the factorization, the SE scheme may prevent future diagonals from growing very large and negative. Overall, these differences in schedule and modification formulas tend to produce, in general, a smaller *variance* of $\|E\|_\infty$ for the SE algorithm for the problems examined: $\|E\|_\infty$ for SE is rarely very large or very small. In particular, the observation that $\overline{M}$ is rarely barely positive definite with SE appears advantageous in our optimization context, especially for highly nonlinear problems. Since it has been suggested that limited-memory quasi-Newton methods perform better than truncated Newton methods for highly nonlinear functions [16], the SE MC may improve the competitiveness of truncated Newton methods for highly nonlinear functions that arise in practice.

Acknowledgments. The author thanks the reviewers and the associate editor, Margaret Wright, for many valuable insights and instructive comments.

REFERENCES

[1] R. Almgren, *Using TNPACK to model crystal growth*, private communication, 1991.
[2] R. S. Dembo and T. Steihaug, *Truncated-Newton algorithms for large-scale unconstrained optimization*, Math. Programming, 26 (1983), pp. 190–212.

[3] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[4] P. DEUFLHARD, *Global inexact Newton methods for very large scale unconstrained nonlinear problems*, in Addendum to the Proceedings of the Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, April 1–5, 1990.

[5] S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ, AND A. H. SHERMAN, *Yale sparse matrix package*, I: *the symmetric codes*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1145–1151.

[6] S. C. EISENSTAT, M. H. SCHULTZ, AND A. H. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM J. Sci. Statist. Comp., 2 (1981), pp. 225–237.

[7] ———, *Efficient implementation of sparse symmetric Gaussian elimination*, in Advances in Computer Methods for Partial Differential Equations, R. Vichnevetsky, ed., AICA, New Brunswick, NJ, 1975, pp. 33–39.

[8] E. ESKOW AND R. B. SCHNABEL, *Software for a new modified Cholesky factorization*, Computer Science Dept. Tech. Rep. CU-CS-443-89, Univ. of Colorado, Boulder, CO, 1989.

[9] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *Computing forward-difference intervals for numerical optimization*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 310–321.

[10] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimizaton*, Academic Press, London, 1983.

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

[12] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Algorithm* 566: *Fortran subroutines for testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 136–140.

[13] ———, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41.

[14] J. J. MORÉ AND D. J. THUENTE, *On line search algorithms with guaranteed sufficient decrease,* Mathematics and Computer Science Division Preprint MCS-P153-0590, Argonne National Laboratory, Argonne, IL, 1990.

[15] S. G. NASH, *Preconditioning of truncated-Newton methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 599–616.

[16] S. G. NASH AND J. NOCEDAL, *A numerical study of the limited memory* BFGS *method and the truncated-Newton method for large-scale optimization*, SIAM J. Optimization, 1 (1991), pp. 358–372.

[17] S. G. NASH AND A. SOFER, *Assessing a search direction within a truncated Newton method*, Oper. Res. Lett., 9 (1990), pp. 219–221.

[18] D. P. O'LEARY, *A discrete Newton algorithm for minimizing a function of many variables*, Math. Programming, 23 (1982), pp. 20–33.

[19] T. SCHLICK, *Modeling and minimization techniques for predicting three-dimensional structures of large biological molecules*, Ph. D. thesis, Dept. of Mathematics, New York Univ., 1987. (Available through University Microfilm International.)

[20] T. SCHLICK, S. FIGUEROA, AND M. MEZEI, *A molecular dynamics simulation of a water droplet by the implicit-Euler/Langevin scheme*, J. Chem. Phys., 94 (1991), pp. 2118–2129.

[21] T. SCHLICK AND A. FOGELSON, *TNPACK—a truncated Newton minimization package for large-scale problems*: (I) *algorithm and usage*, ACM Trans. Math. Software, 18 (1992), pp. 46–70.

[22] ———, *TNPACK—a truncated Newton minimization package for large-scale problems*: (II) *implementation examples*, ACM Trans. Math. Software, 18 (1992), pp. 71–111.

[23] T. SCHLICK, B. E. HINGERTY, C. S. PESKIN, M. L. OVERTON, AND S. BROYDE, *Search strategies, minimization algorithms, and molecular dynamics simulations for exploring conformational spaces of nucleic acids*, in Theoretical Biochemistry and Molecular Biophysics: A Comprehensive Survey, D. L. Beveridge and R. Lavery, eds., Adenine Press, Guilderland, New York, 1991, pp. 39–58.

[24] T. SCHLICK AND M. OVERTON, *A powerful truncated Newton method for potential energy minimization*, J. Comput. Chem., 8 (1987), pp. 1025–1039.

[25] R. B. SCHNABEL AND E. ESKOW, *A new modified Cholesky factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1136–1158.

[26] X. ZOU, I. M. NAVON, M. BERGER, P. K. H. PHUA, T. SCHLICK, AND F. X. LEDIMET, *Numerical experience with limited-memory quasi-Newton and truncated Newton methods*, SIAM J. Optimization, 3 (1993), to appear.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 446–460, March 1993

OPTIMAL PARALLEL SOLUTION OF SPARSE TRIANGULAR SYSTEMS*

FERNANDO L. ALVARADO[†] AND ROBERT SCHREIBER[‡]

Abstract. This paper considers the parallel solution of a sparse system $Lx = b$ with triangular matrix L, which is often a performance bottleneck in parallel computation. When many systems with the same matrix are to be solved, the parallel efficiency can be improved by representing the inverse of L as a product of a few sparse factors. The factorization with the smallest number of factors is constructed, subject to the requirement that no new nonzero elements are created. Applications are to iterative solvers with triangular preconditioners, to structural analysis, or to power systems applications. Experimental results on the Connection Machine show the method to be highly valuable.

Key words. sparsity, sparse matrices, numerical linear algebra, triangular matrices, iterative methods, parallel computation

AMS(MOS) subject classifications. 65F05, 65F10, 65F50, 65Y05

1. Introduction.

Consider the solution of a sparse, nonsingular, lower-triangular linear system $Lx = b$ in a parallel environment. (All our results extend in a trivial way to upper-triangular systems, a fact we shall not mention again.) We consider the case where the problem must be solved for multiple right-hand side vectors b, and these vectors are not available all at once. By preprocessing, we shall reduce the number of parallel steps required.

Important applications where multiple right-hand sides arise include finite element applications, preconditioned iterative solvers for linear systems, solution of initial value problems by implicit methods, and variants of Newton's method for the solution of nonlinear equations.

There are two possible approaches to the parallel solution of triangular systems of equations. The usual approach is to exploit whatever parallelism is available in the usual substitution algorithm [4], [10]. The second, which requires preprocessing, works with some representation of L^{-1}. In sequential sparse matrix computation, substitution is universally favored because it retains the sparsity of the problem [7], while L^{-1} is usually much denser than L [8].

Here we consider a factorization

$$L^{-1} = \prod_{k=1}^{m} Q_k$$

with sparse factors. Such a factorization is possible in which the factors have no more nonzeros than L [2], [3], [8]. The chief advantage of a factorization of L^{-1} is that all the necessary multiplications for the computation of $Q_k x$ can be performed concurrently. The necessary additions can be done in time logarithmic in the largest

* Received by the editors July 31, 1990; accepted for publication (in revised form) March 30, 1992.

† The University of Wisconsin, Madison, Wisconsin 53706. This work was supported in part under National Science Foundation contracts ECS-8822654 and ECS-8907391.

‡ Research Institute for Advanced Computer Science, MS T045-1, NASA Ames Research Center, Moffett Field, California 94035. This work was supported by the Numerical Aerodynamic Simulation Systems Division of NASA and by the Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association, and was partly accomplished during a visit to the European Center for Research and Training in Scientific Computing (CERFACS).

number of nonzeros in a row of Q_k. Thus it is possible to take advantage of more parallelism in the solution of these equations. The remainder of this introduction reviews the use of partitioned inverses of L.

Any triangular matrix L can be expressed as a product

$$L = L_1 L_2 \cdots L_n,$$

where

$$\begin{aligned} (L_k)_{ik} &= L_{ik} \quad \text{for} \quad k \le i \le n, \\ (L_k)_{jj} &\equiv 1 \quad \text{for} \quad j \ne k, \\ (L_k)_{ij} &\equiv 0 \quad \text{otherwise}. \end{aligned}$$

The matrices L_k are known as elementary lower-triangular matrices. They can be grouped into several factors,

$$L = \prod_{k=1}^{m} P_k, \tag{1}$$

where

$$P_k = L_{e_{k-1}+1} L_{e_{k-1}+2} \cdots L_{e_k} \tag{2}$$

and

$$0 = e_0 < e_1 < \cdots < e_m = n. \tag{3}$$

Here $\{e_k\}_{k=0}^{m}$ is a monotonically increasing integer sequence. The factor P_k is lower triangular and is zero below its diagonal in all columns except columns $e_{k-1} + 1$ through e_k, where it is identical to L. Consider, for example,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 2 & 3 & 0 \\ 4 & 0 & 1 & 1 \end{bmatrix}.$$

By choosing $e_0 = 0, e_1 = 2$, and $e_2 = 4$, L is partitioned as

$$L = (L_1 L_2)(L_3 L_4) = P_1 P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

It follows from (1) that

$$x = L^{-1} b = \prod_{k=m}^{1} P_k^{-1} b. \tag{4}$$

Thus, we may compute x as follows:

```
$x = b$;
for $k = 1$ to $m$ do
    $x = P_k^{-1} x$;
od;
```

1.1. Problems addressed. We say that the matrix X is *invertible in place* if $X_{ij} = 0 \Rightarrow (X^{-1})_{ij} = 0$. Elementary lower-triangular matrices are invertible in place. Thus every lower-triangular matrix has at least one partition (1) with factors that invert in place.

DEFINITION 1. A partition (1) in which the factors P_k are invertible in place is called a no-fill partition. A no-fill partition of L with the smallest possible number of factors is a *best no-fill partition.*

Let $G(L)$ be a digraph with vertices $V = \{1, 2, \ldots, n\}$ and directed edges $E = E(L) \equiv \{(j,i) \mid L_{ij} \neq 0\}$. Think of the edge (j,i) as an arrow going from vertex j to vertex i. If L is lower triangular, then for all edges (j,i) we have $i > j$. $G(L)$ is therefore an acyclic digraph.

Consider the matrix L with graph $G(L)$ illustrated in Fig. 1. L has a best no-fill partition

$$L = (L_1)(L_2)(L_3L_4)(L_5)(L_6L_7).$$

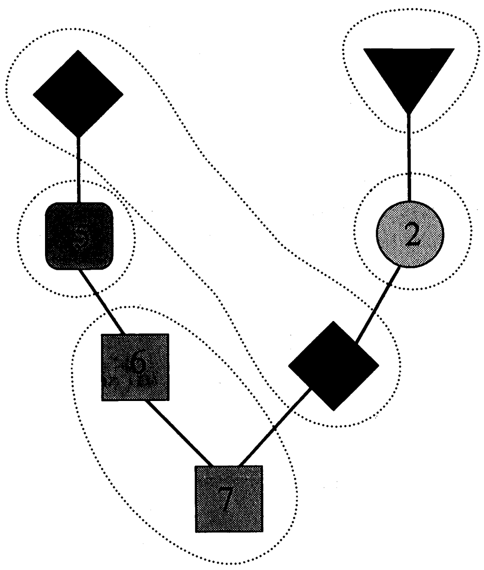


FIG. 1. *Best no-fill partition for a graph without reordering. Five factors are required.*

This partition has five factors. It is possible to symmetrically permute the rows and columns of L such that L remains lower triangular and $G(L)$ is as illustrated in Fig. 2. A best no-fill partition of this reordered L is

$$L = (L_1L_2)(L_3L_4)(L_5L_6L_7),$$

which has only three factors.

DEFINITION 2. A *best reordered partition* of L is a symmetric permutation of the rows and columns of L such that the permuted matrix is triangular and has a best no-fill partition with the fewest possible factors.

The aims of this work, and an outline of the paper, are as follows. First, we develop a theory of efficient algorithms for computing best no-fill and best reordered

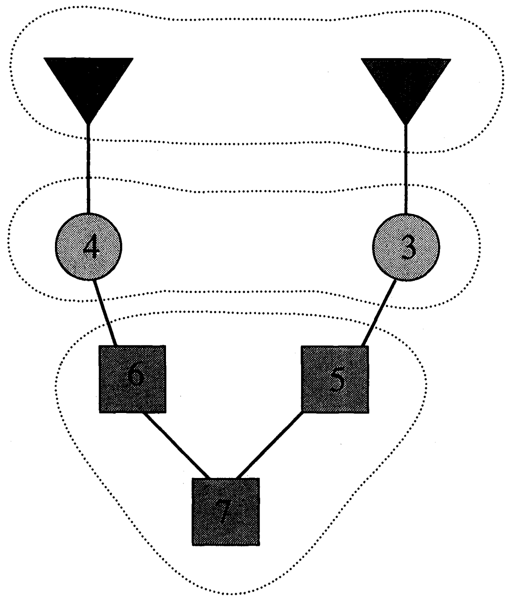


FIG. 2. *Best no-fill partition for a reordered graph. Only three factors are required.*

partitions. We give a proof of correctness for the procedure of Alvarado, Yu, and Betancourt for best no-fill partitioning in §2. In §3 we give an efficient procedure (and proof) for best reordered partitioning. Second, we determine how useful these ideas are in practice by means of some experiments (§4). Our conclusions are in §5.

We allow arbitrary lower-triangular matrices. Pothen and Alvarado [13] give a simplified version of our methods for the case of Cholesky factors (they exploit the fact that the graph of a Cholesky factor is chordal) that has $O(n)$ complexity.

Efficient methods for Cholesky factorization exploit cliques of vertices having the same adjacency, all of which are eliminated as a group; these vertex subsets are called supernodes [5]. It is possible to take advantage of the dense submatrices of L corresponding to supernodes in both substitution and partitioned inverse methods. Our results are stronger; all members of a supernode are included in the same factor in a best no-fill partition, but the converse is not true.

1.2. Terminology. A *topological ordering* of an acyclic digraph is a numbering of vertices in which $(j,i) \in E$ implies $j < i$. Every acyclic digraph has topological orderings. The digraph of every triangular matrix is acyclic; moreover, its conventional ordering is topological. If $G(L)$ is reordered so that vertex i is numbered $\alpha(i)$ and the new ordering is topological, then the symmetric permutation of rows and columns that moves row i to row $\alpha(i)$ leaves L triangular.

For $(j,i) \in E$ we say that j is a *predecessor* of i and i is a *successor* of j.

The *indegree* of a vertex i is the number of vertices j such that $(j,i) \in E$. The indegree of vertex i is the number of nonzeros in the ith row of L, which is the number of predecessors of i.

Let $G = G(L)$. For $j \in V$, madj(j) is the set of higher numbered neighbors of j, i.e., the set $\{i > j \mid L_{ij} \neq 0\}$. The set madj$(j)$ is the set of rows that are nonzero in the jth column of L, which is the set of successors of j.

The *transitive closure* of a digraph $G = (V, E)$ is the graph $G' = (V, E')$ where $E' = \{(j, i) \mid \text{there is a } j \to i \text{ path in } G\}$. The digraph $G = (V, E)$ is *transitively closed* if it is equal to its own transitive closure.

LEMMA 1 (Gilbert [9]). *If L is a nonsingular lower-triangular matrix, then $G(L^{-1})$ is the transitive closure of $G(L)$.*

DEFINITION 3. Let $G = G(L) = (V, E)$ be an acyclic digraph. Given a subset $S \subseteq V$, define the *column subgraph* $G_S = (V, E_S)$ (where $E_S \equiv \{(j, i) \in E | j \in S\}$); this is the graph of the lower-triangular matrix obtained by zeroing all columns of L not in S.

The proof of the following result follows immediately from Lemma 1.

THEOREM 2. *Let a partition* (3) *and corresponding factorization* (1) *be given. The factors P_k are invertible in place if and only if each column subgraph $G(P_k) = G_{\{e_{k-1}+1,\ldots,e_k\}}$ is transitively closed.*

We write $\eta(X)$ for the number of nonzeros in the matrix X.

2. Best no-fill partitioning.

The following algorithm was proposed by Alvarado, Yu, and Betancourt (who call it PA2) [3].

ALGORITHM P1.
Input: $L = L_1 L_2 \cdots L_n$.
Output: A best no-fill partition of L.

```
$i \leftarrow 1$;   $k \leftarrow 1$;
while ($i \leq n$) do
    (Find the largest integer $r \geq i$ such that $L_i \cdots L_r$ is invertible in place.)
    $r \leftarrow i$;
    while every successor of $r$ is a successor of all the predecessors of $r$ do
        $r \leftarrow r + 1$;
    od
    $P_k \leftarrow L_i \cdots L_r$;
    $k \leftarrow k + 1$;   $i \leftarrow r + 1$;
od
```

Alvarado, Yu, and Betancourt did not mention the issue of optimality. Here we show that Algorithm P1 determines a best no-fill partition.

LEMMA 3. *If $L_1 \cdots L_r$ is invertible in place, then $L_2 \cdots L_r$ is, too.*

Proof. The graph obtained by removing a source vertex and its incident edges from a transitively closed graph is transitively closed. The lemma thus follows from Lemma 1.

THEOREM 4. *Algorithm* P1 *produces a best no-fill partition of L.*

Proof. We use induction on the number of columns of L that are nonzero below the diagonal. The base of the induction is provided by the trivial case $L = I$. We assume that the result is true if the first column of L is zero below the diagonal, and then show it to be true for any lower-triangular matrix L.

Suppose that Algorithm P1 produces a partition $L = P_1 \cdots P_m$. Let $Q \equiv P_2 \cdots P_m$. Clearly, there does not exist a no-fill partition with e_1 any larger than that produced by Algorithm P1. Let $L = \hat{P}_1 \cdots \hat{P}_{m'}$ be a different no-fill partition. Suppose that $P_1 = \hat{P}_1$. By the inductive hypothesis, Algorithm P1, when applied to Q, produces a best no-fill partition, so at least $m - 1$ factors are required in any no-fill partition of Q. Thus $m' \geq m$.

On the other hand, perhaps $\hat{P}_1 = L_1 \cdots L_{e_1'}$ with $e_1' < e_1$, i.e., $\hat{P}_1$ has *fewer* nonzero columns than does P_1. The matrix $\hat{Q} \equiv \hat{P}_2 \cdots \hat{P}_{m'}$ has a no-fill partition with $(m' - 1)$ factors. By the previous lemma, we may remove the leftmost elementary lower-triangular factor of $\hat{Q}$ and still have an $m' - 1$ factor no-fill partition. Continuing in this way we find such a partition of Q. But by the inductive hypothesis, any no-fill partition of Q has at least $m - 1$ factors. Thus we again see that $m' \geq m$. □

Best no-fill partitions are not unique. For example,

$$L = \begin{bmatrix} x & & & \\ 0 & x & & \\ x & x & x & \\ 0 & x & x & x \end{bmatrix}$$

has best no-fill partitions

$$L = L_1(L_2L_3L_4)$$

and

$$L = (L_1L_2)(L_3L_4).$$

Let $G = (V, E)$ be an acyclic digraph. Define level$(i), i \in V$ to be the length of the longest path from a source vertex to vertex i. The computation of levels is straightforward, requiring $O(\eta(L))$ time. Any parallel implementation of a substitution method for solving $Lx = b$ requires at least as many parallel steps as there are levels in $G(L)$.

3. Best reordered partitioning.

This section describes two new algorithms for best reordered partitioning. Each algorithm finds a topological ordering of an acyclic digraph G such that the reordered graph has a best no-fill partition with the smallest possible number of factors. The first algorithm (RP1; see Fig. 3) is a straightforward but inefficient "greedy" algorithm. We present it in order to motivate a better algorithm, RP2.

Consider the following graph model of best reordered partitioning. Let $G = G(L) = (V, E)$. We seek a partition $V = \bigcup_{k=1}^{m} S_k$. We number the vertices within a subset S_k consecutively. We reorder rows and columns of L according to this vertex numbering, and then partition L as in (3). The factors P_k and the subsets S_k will correspond; that is, the columns of P_k that are nonzero below the diagonal will be those corresponding to the vertices in S_k. We call the subsets S_k in this partition *factors* in analogy with their corresponding factors P_k of L.

In order to have factors P_k that invert in place, we require that the column subgraphs G_{S_k} be transitively closed (Theorem 2). Second, in order for the new vertex ordering to be topological, we require that the partition of V be consistent with the partial order E: if $r > s$, then there is no edge in E from a member of S_r to a member of S_s. We produce the required topological ordering of $G(L)$ starting with a topological ordering of S_1, then of S_2, and so on.

```
ALGORITHM RP1 (Reorder, Permute 1).
Input: An acyclic digraph $G(L)$.
Output: A permutation $\alpha : V \longrightarrow \{1, \ldots, n\}$ and a partition of $L$.
Compute level$(v)$ for all $v \in V$;
```

```
max-level $\leftarrow \max_{v\in V}(\text{level}(v))$;
$i \leftarrow 0$;   $k \leftarrow 1$;   $e_0 \leftarrow 0$;
**while** $i < n$ **do**
    $S_k \leftarrow \emptyset$;   $e_k \leftarrow i$;
    $\ell \leftarrow \min\{j \mid$ there is an unnumbered vertex at level $j\}$;
    **repeat**
        **for** every vertex $v$ at level $\ell$ **do**
            **if** (([Condition 1] $v$ is unnumbered) **&&**
                ([Condition 2] Every predecessor of $v$ has been numbered) **&&**
                ([Condition 3] Every successor of $v$ is a successor of all
                      $u \in S_k$ such that $u$ is a predecessor of $v$)) **then**
                $i \leftarrow i+1$;   $\alpha(v) \leftarrow i$;
                $S_k \leftarrow S_k \cup \{v\}$;   $e_k \leftarrow e_k + 1$;
            **fi**
        **od**
        $\ell \leftarrow \ell + 1$;
    **until** $\ell >$ max-level or no vertices at level $\ell - 1$ were included in $S_k$;
    $P_k \leftarrow L_{e_{k-1}} \cdots L_{e_k}$;   $k \leftarrow k+1$;
**od**
```

Figure 3 illustrates the reason for Condition 3. Different node shapes are used to denote different factors. It is possible to combine nodes 1, 2, and 5 into the same factor according to Condition 3: all the successors of 5 are successors of the predecessors of 5. It is *not* possible, however, to combine node 6 into the same factors as nodes 3 and 4 because this rule is violated. Figure 4 illustrates a case where it would be possible to merge node 5 into S_1, along with nodes 2 and 3, without violating the requirement of transitive closure: Condition 3 is satisfied by 5. But, since one of the predecessors of 5 is not yet numbered when node 5 is considered, it cannot be included, since the resulting ordering would then fail to be topological. Condition 2 does not hold. Thus node 5 must be in a different factor.

THEOREM 5. *Algorithm* RP1 *finds a best-reordered partition.*

Proof. The proof is by induction, as is the proof of Theorem 4. Consider any permutation Γ and partition $L_\Gamma = \Gamma L \Gamma^T = \prod_{k=1}^{m} P_k$ with P_k defined by (2) and (3), and with P_k invertible in place. We claim that the partition point e_1 is no larger than that produced by RP1. For it follows from the construction of P_1 that any vertex of $G(L)$ not included in S_1 by RP1 either has an unnumbered predecessor, or else its inclusion renders G_{S_1} not transitively closed. Hence the subset S_1 selected by RP1 is maximal. Now, by the inductive hypothesis, RP1 partitions $G \setminus S_1$, using $m-1$ invertible-in-place factors. Moreover, by the argument used to prove Lemma 3, a graph obtained by adding source vertices to $G \setminus S_1$ requires at least $m-1$ factors in any optimal partition. Thus if we take a subset $\hat{S}_1$ of S_1 as the first factor, we can achieve nothing better. $\square$

We would like to ensure that running time is bounded by a small constant multiple of $\eta(L)$. But the running time of Algorithm RP1 is large in some cases. Consider a dense lower-triangular matrix of order n. RP1 takes $O(n^3)$ time in this case, since the cost of checking whether all successors of vertex j are also successors of its predecessors is $O(j(n-j))$.

We now introduce two new data structures in order to improve the efficiency of RP1. We can improve the performance of RP1 (to $O(n^2)$ for a dense matrix) by

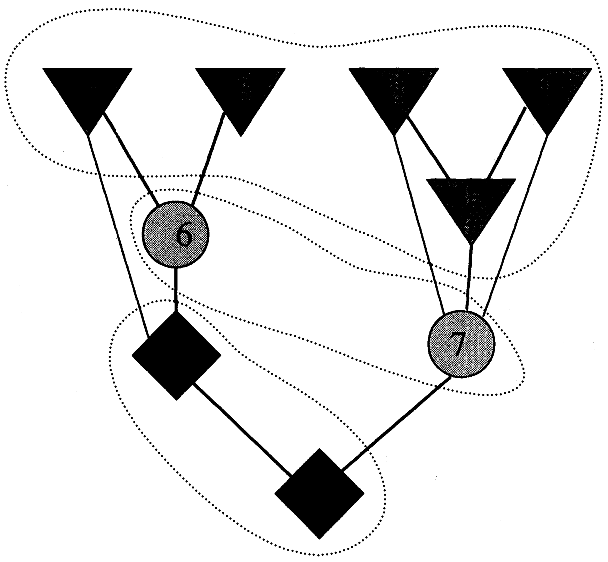


FIG. 3. *Illustration of Condition* 3. *Successors of predecessors of a node must be successors of the node itself. Node* 5 *can be included in the same factor as nodes* 1 *and* 2, *but node* 6 *cannot be included with* 3 *and* 4. *Factors are denoted by shapes and shading.*

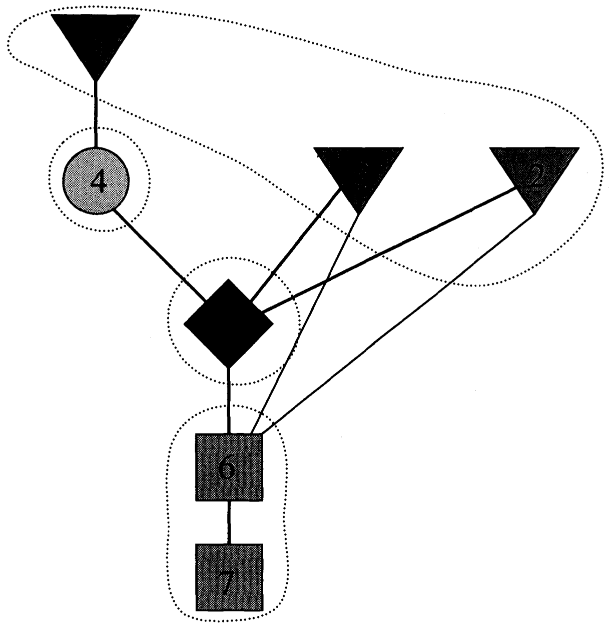


FIG. 4. *Illustration of Condition* 2. *A node must be excluded from a factor because some of its predecessors are not numbered. One predecessor of node* 5 *(node* 4*) has not been numbered by the time node* 5 *becomes eligible to join the factor containing nodes* 2 *and* 3.

ensuring that a vertex is not examined for possible inclusion in S_k until all of its predecessors have been numbered. To do so, for every vertex we count the number of its unnumbered predecessors. Initially, this is its indegree. When the count reaches zero we can consider the vertex for inclusion in a factor. This is a standard technique for finding topological orderings [11].

We can avoid much of the work associated with the checking at Condition 3 of RP1. Let u and v be numbered vertices, both of which have been included in the current factor S_k. Assume that v is a successor of u. Then we must have that $\text{madj}(v) \subseteq \text{madj}(u)$, otherwise v would have failed the test at Condition 3. Thus we need not consider vertex u when applying the requirements of Condition 3 to a vertex that is also a successor of v. We make use of this in the faster implementation (RP2; see Fig. 3) by keeping track of the set of predecessors of each vertex that may need to be examined in checking Condition 3. In the situations above, when v is included in S_k we remove u from the predecessor sets of v's successors, thus avoiding the unnecessary checking.

ALGORITHM RP2 (Reorder, Permute 2).
Input: An acyclic digraph $G(L)$.
Output: A permutation *and* a partition of L.

```
forall $v \in V$ do
    $\text{pred}(v) \leftarrow \{u \mid L_{vu} \neq 0\}$;
    $\text{count}(v) \leftarrow \text{indegree}(v)$;
    Compute $\text{level}(v)$;
od
$\text{max-level} \leftarrow \max_{v \in V}(\text{level}(v))$;
$i \leftarrow 0$;    $k \leftarrow 1$;    $e_0 \leftarrow 0$;
$E \leftarrow \{v \in V \mid \text{count}(v) = 0\}$;
while $i < n$ do
    $S_k \leftarrow \emptyset$;    $e_k \leftarrow i$;
    $\ell \leftarrow \min\{j \mid$ there is an unnumbered vertex at level $j\}$;
    repeat
        for every vertex $v \in E$ at level $\ell$ do
            if ([Condition 3′] Every successor of $v$ is a successor of all
                        $u \in \text{pred}(v)$) then
                $i \leftarrow i + 1$;    $\alpha(v) \leftarrow i$;
                $S_k \leftarrow S_k \cup \{v\}$;    $e_k \leftarrow e_k + 1$;
                for every successor $w$ of $v$ do
                    $\text{pred}(w) \leftarrow \text{pred}(w) \setminus \text{pred}(v)$;
                    $\text{count}(w) \leftarrow \text{count}(w) - 1$;
                    if $\text{count}(w) = 0$ then $E \leftarrow E \cup \{w\}$;  fi
                od
            fi
        od
        $\ell \leftarrow \ell + 1$;
    until $\ell >$ max-level or no vertices at level $\ell - 1$ were included in $S_k$;
    $P_k \leftarrow L_{e_{k-1}} \cdots L_{e_k}$;    $k \leftarrow k + 1$;
od
```

The computation of $\text{pred}(v)$ is straightforward, requiring $O(\eta(L))$ time. Clearly, the innermost loop of Algorithm RP2 is executed $\eta(L)$ times. For we have that the sum of count(v) over all vertices v is just $\eta(L)$, and this sum decreases by one for every execution of this innermost loop. The other statements in the scope of the **then** clause are executed no more than n times. It is still possible that testing Condition $3'$ will be costly. Indeed, we can construct examples for which the running time is greater than $\eta(L)$, but in practice this is unlikely to happen.[1]

4. Examples. This section gives several examples of the performance of the proposed algorithms. Several examples compare P1 and RP1 with respect to the number of factors in the partitions they find. We also examine the effect of the initial ordering of rows and columns of a matrix A on the number of factors in a best reordered partition of its Cholesky or incomplete Cholesky factor L. An experiment shows that on the Connection Machine, the solution process (4) can be faster than substitution by orders of magnitude. (The experiments and illustrations were done with the aid of Alvarado's Sparse Matrix Manipulation System [1].)

First, we compare Algorithms P1 and RP1. Table 1 uses five power system matrices ranging in size from 118 to 1993. Table 2 gives results for matrices arising from five-point finite difference discretizations.

TABLE 1
Effect of primary ordering on number of factors in best no-fill partitions (P1) *and best reordered partitions* (RP1) *for power system matrices.*

	Min. degree		MMD		MLMD	
Size	P1	RP1	P1	RP1	P1	RP1
118	53	14	10	10	6	5
352	132	21	13	12	8	8
707	213	26	23	18	11	10
1084	309	26	33	24	14	11
1993	563	35	41	25	15	15

TABLE 2
Effect of primary ordering on number of factors in best no-fill partitions (P1) *and best reordered partitions* (RP1) *for five-point finite difference operators on grid graphs.*

	Min. deg.		MMD		MLMD	
Grid size	P1	RP1	P1	RP1	P1	RP1
5 by 5	10	7	4	4	4	4
10 by 10	21	13	10	10	7	6
15 by 15	33	18	12	12	6	6
20 by 20	32	17	19	18	13	9
25 by 25	27	19	18	17	12	8
30 by 30	36	18	23	17	17	10
35 by 35	26	18	22	19	19	11
40 by 40	50	23	27	22	20	11

In each case, the original coefficient matrix is first ordered, then the structure of its Cholesky factor L is found. We need to distinguish this first fill-reducing ordering of A from the reordering of L found by RP1. We call the ordering of A the primary ordering. Three primary ordering procedures are used: the minimum degree algorithm [14], the multiple minimum degree (MMD) algorithm [12], and the minimum level, minimum degree (MLMD) algorithm [6].

[1] Consider a graph with $3k$ vertices arranged in three levels of k each. All vertices at level ℓ are adjacent to all at level $\ell + 1$, for $\ell = 1, 2$. Running time is $O(\eta(L)^{3/2})$ again.

For each matrix and primary ordering algorithm, two partitioning methods are compared: Algorithm P1, which simply partitions L optimally without reordering it, and Algorithm RP1 which reorders the matrix and generates an optimal partition. In most cases, Algorithm RP1 gives a smaller number of factors than PA1, while in a few cases both algorithms give the same number of factors.

We observe that RP1 reduces the number of factors at no expense in added fills. Its effect is most dramatic if the underlying primary ordering is the minimum degree algorithm. On the other hand, the best results are obtained when the MLMD algorithm is used for the primary ordering, even though the relative improvement attainable by Algorithm RP1 over Algorithm P1 is small. The results for the MMD algorithm fall somewhere between minimum degree and MLMD. The reduction in the number of factors achieved by RP1 in comparison with P1 is quite dramatic when MMD is the primary ordering. (MLMD tends to produce more fill than MMD, so the question of which is preferable is not simple.)

Figures 5 and 6 provide a different illustration of the effect of primary ordering on matrix structure and partitioning. The matrix L is the Cholesky factor for a 10×10 finite difference grid.

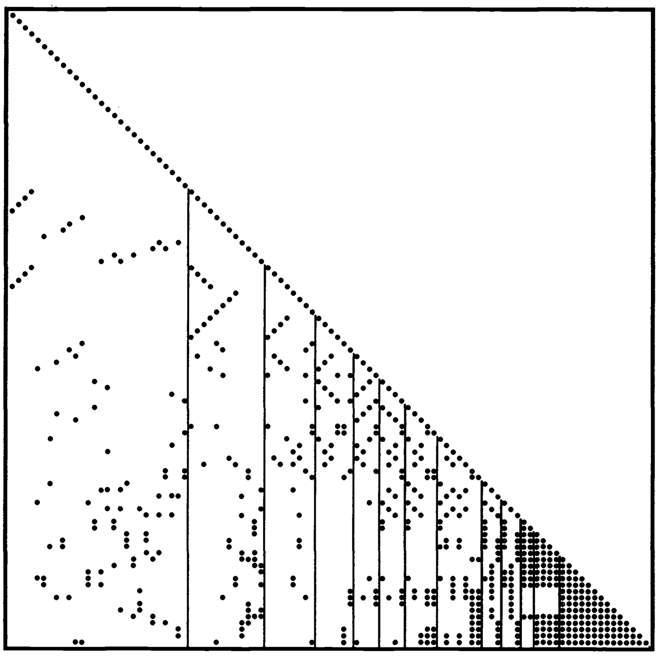


FIG. 5. *Five-point finite difference matrix for* 10 *by* 10 *grid. Matrix ordered by the minimum degree algorithm, then reordered and partitioned by* RP1. *Thirteen factors result.*

The second experiment we report compares the solution procedure (4) with a data-parallel forward substitution method on the Connection Machine model CM-2, a highly parallel single instruction multiple data (SIMD) computer.

An efficient, parallel substitution algorithm is implemented in CM Fortran, a dialect of Fortran 90.

The data structure consists of several arrays of length equal to $\eta(L)$. We associate, at least conceptually, a set of $\eta(L)$ virtual processors, one with each position in these

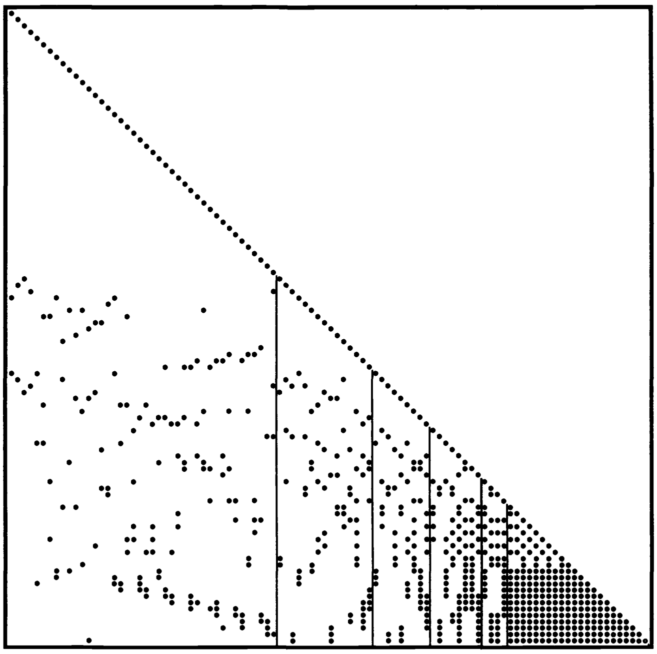


FIG. 6. *Five-point finite difference matrix for* 10 *by* 10 *grid. Matrix ordered by* MLMD, *then reordered and partitioned by* RP1. *Six factors result.*

arrays. We choose to store the factors $\hat{L}$ and D where $L = \hat{L}D$ and $\hat{L}$ is unit triangular, and solve $Lx = b$ via $\hat{L}x = D^{-1}b$, since this removes a multiplication from the inner loop. The matrix $\hat{L}$ is stored as a one-dimensional array containing its nonzeros in column-major order. In addition, the level of vertex j in $G(L)$ is stored along with $\hat{L}_{ij}$. The elements of b are stored at the processors containing the diagonal of $\hat{L}$. The solution x overwrites b. Finally, a boolean vector indicates the location of the diagonal elements in the arrays. This vector thus segments the arrays into columns of differing lengths; in other words, the matrix is stored as a ragged array of columns of nonzeros. The Connection Machine software provides some operations for such data structures. It allows broadcast of values from diagonal elements to all elements of the corresponding column (called a segmented copy scan) and summation of the values in a column (with a segmented add scan). Also, the Connection Machine router allows processors to send data to any other processor or read from any other processor; this is expressed using a vector-valued subscript in CM Fortran. Calls to utility library routines were used for the scan operations, which are not part of CM Fortran.

The algorithm loops sequentially over levels of $G(L)$, starting with the sources (level zero). At the beginning of step ℓ, those elements x_j for which level$(j) = \ell$ are known. Recall that x_j is stored in the processor holding $\hat{L}_{jj}$. These known values of x are sent to the processors holding the corresponding column of $\hat{L}$ by a segmented copy scan. These processors then multiply their $\hat{L}$ value by the element of x they receive. The router is then used to permute all these products into row-major order, so that the elements of each set $R_i \equiv \{\hat{L}_{ij}x_j \mid L_{ij} \neq 0 \text{ and level } (j) = \ell\}$ are stored in consecutive locations. The vector-valued subscript used to accomplish this permutation is computed in a preprocessing step. The rows R_i are now a ragged array. A segmented add scan forms the sums of these partial results within rows. Finally, the router is used to send the sum of the elements of R_i to the processor holding $\hat{L}_{ii}$ and b_i where it can be subtracted from b_i. (In fact, our code avoids this last subtraction entirely, doing it as part of the add scan above.)

Thus, an iteration of the loop involves one parallel multiplication, one copy scan and one add scan, and two uses of the router for permutation of data. The time required to set up and load the data structure, including the computation of the permutation used, the loading of b, and the extraction of x, was not timed.

The sequential algorithm for solving a triangular system looks nearly identical to the code for matrix–vector products with a triangular matrix. Thus it may come as no surprise that our partitioned solution method in CM Fortran is nearly identical to our substitution method. The inner loop involves the same operations. But the number of executions of the loop is equal to the number of factors in the partition of L rather than the number of levels in its graph.

We report the CM performance of these two methods for an $n \times n$, dense, unit lower-triangular matrix L. Our results are obtained with CM Fortran in the "slicewise" execution model, which treats each Weitek chip of the CM-2 as a processor. For this experiment we used 256 Weitek processors on the Connection Machine at NASA Ames. Results are given in Table 3. Clearly, the partitioned method is superior, by a factor roughly equal to the ratio of the number of levels in $G(L)$ (n) to the number of factors (one) in the partition of L.

TABLE 3
CM-2 *times for full matrix substitution and partitioned solution.*

n	Nonzeros	Levels in $G(L)$	Substitution time	Factors	Partitioned soln. time
256	32,896	256	7.34 secs	1	0.04 secs
512	131,328	512	50.22 secs	1	0.21 secs

Next, we performed an experiment using several large, sparse, unit lower-triangular matrices. We begin with a sparse matrix A of order 4037, obtained from a triangular mesh in the region around a three-element airfoil. Three matrices L_1, L_2, and L_3 are obtained by approximate factorization.

L_1 is obtained by an incomplete LU factorization of A; we carry out the Gaussian elimination process, but we allow nonzeros in L (and U) only where there is a nonzero in A^2. The ordering of A is obtained from a lexicographic sort of the (x, y) coordinates of the grid which leads to the matrix; this ordering produces a large number of levels in $G(L)$.

L_2 is the incomplete LU factor obtained when a variant of MLMD is used as the primary ordering of A.

L_3 is the exact lower-triangular factor of A, with the same primary ordering as for L_2.

In Table 4 we give the size of these factors, the number of levels, which is proportional to the time required for our parallel substitution algorithm, and the number of partitions, which is in practice proportional to the time required by the partitioned solution algorithm (4).

These results confirm that the time required to solve a triangular system by partitioning of the inverse is quite well predicted by the number of factors in the partition. It has also shown that the number of levels in $G(L)$ is also a good predictor

TABLE 4
CM-2 *times for sparse triangular substitution and partitioned solution.*

Matrix	Ordering	Factorization	Nonzeros	Levels in $G(L)$	Substitution time	Factors	Partitioned soln. time
L_1	RCM	ILU	23,526	823	19.22 secs	816	11.66 secs
L_2	MLMD	ILU	26,793	78	1.38 secs	66	1.20 secs
L_3	MLMD	exact	118,504	311	16.78 secs	16	0.87 secs

of the time required for solution by substitution methods. We see that when L has a fairly rich structure there is a great advantage to the use of the partitioned method, but when L is very sparse there is little gained. The use of an MLMD primary ordering improves both substitution and partitioned methods. However, with the introduction of the additional fill in the exact factor L_3 (compared with L_2), the number of levels in $G(L)$ increases sharply (as does the time for substitution) while the number of factors in the best reordered partition drops dramatically. The difference in the solution time, even for this problem of modest size, is about a factor of twenty. Thus we conclude that the method can be quite useful in highly parallel machines when the matrix L has a rich enough structure, as happens when it is an exact triangular factor.

5. Conclusions. An algorithm for no-fill partitioning of lower-triangular matrices with the fewest possible factors has been presented and proven to be optimal.

We have done a number of experiments with the Cholesky factors of sparse matrices, A. The structure of these factors is influenced by the ordering of rows and columns of A. (This ordering is chosen to reduce fills during the factorization process.) This primary ordering also influences the benefits attainable by the proposed algorithm. If L is constructed with the minimum degree algorithm, the best no-fill partition of L (without a change of ordering) tends to have many more factors than the best reordered partition developed here. This number can be further reduced if an MLMD primary ordering is used.

An experiment on the Connection Machine has shown that solution time is roughly proportional, for fixed nonzero count, to either the number of factors or the number of levels in $G(L)$. Finally, we have seen that for Cholesky factors, a best reordered partition can have far fewer factors than there are levels, so that the method is highly valuable. We conjecture that for sufficiently filled incomplete factors there will be some benefit as well, although for very sparse incomplete factors the improvement is quite modest.

Acknowledgments. The assistance of CERFACS and its staff during the author's visit in August 1989 is gratefully acknowledged.

REFERENCES

[1] F. L. ALVARADO, *Manipulation and visualization of sparse matrices*, ORSA J. Comput., 2 (1990), pp. 180–207.

[2] F. L. ALVARADO, D. YU, AND R. BETANCOURT, *Ordering schemes for partitioned sparse inverses*, SIAM Symp. Sparse Matrices, Salishan Lodge, Gleneden Beach, OR, May 22–24, 1989.

[3] ———, *Partitioned sparse* A^{-1} *methods*, IEEE Trans. Power Systems, 5 (1990), pp. 452–459.

[4] E. ANDERSON, *Parallel implementation of preconditioned conjugate gradient methods for solving sparse systems of linear equations*, Tech. Rep. 805, Center for Supercomputer Research and Development, Univ. of Illinois, Champaign, IL, 1988.

[5] C. ASHCRAFT AND R. GRIMES, *The influence of relaxed supernode partition on the multifrontal method*, ACM Trans. Math. Software, 15 (1989), pp. 291–309.

[6] R. BETANCOURT, *An efficient heuristic ordering algorithm for partial matrix refactorization*, IEEE Trans. Power Systems, 3 (1988), pp. 1181–1187.

[7] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, U.K., 1986.

[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. Rep. 86–750, Cornell University, Ithaca, NY, 1986; SIAM J. Matrix Anal. Appl., 15 (1994), to appear.

[10] S. W. HAMMOND AND R. SCHREIBER, *Efficient* ICCG *on a shared memory multiprocessor*, Internat. J. High-Speed Comput., 4 (1992), pp. 1–21.

[11] E. HOROWITZ AND S. SAHNI, *Fundamentals of Data Structures*, Computer Science Press, New York, 1982.

[12] J.W. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[13] A. POTHEN AND F. L. ALVARADO, *A fast reordering algorithm for parallel sparse triangular solution*, Tech. Rep. CS-91-07, The Pennsylvania State Univ., University Park, PA, 1991, pp. 302–325.

[14] W. F. TINNEY AND J. W. WALKER, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. IEEE, 55 (1967), pp. 1801–1809.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 461–469, March 1993

A FLEXIBLE INNER-OUTER PRECONDITIONED GMRES ALGORITHM*

YOUCEF SAAD†

Abstract. A variant of the GMRES algorithm is presented that allows changes in the preconditioning at every step. There are many possible applications of the new algorithm, some of which are briefly discussed. In particular, a result of the flexibility of the new variant is that any iterative method can be used as a preconditioner. For example, the standard GMRES algorithm itself can be used as a preconditioner, as can CGNR (or CGNE), the conjugate gradient method applied to the normal equations. However, the more appealing utilization of the method is in conjunction with relaxation techniques, possibly multilevel techniques. The possibility of changing preconditioners may be exploited to develop efficient iterative methods and to enhance robustness. A few numerical experiments are reported to illustrate this fact.

Key words. Krylov subspace methods, GMRES, non-Hermitian systems, preconditioned conjugate gradient, variable preconditioners

AMS(MOS) subject classification. 65F10

1. Introduction.

Krylov subspace techniques have increasingly been viewed as general-purpose iterative methods, especially since the popularization of preconditioning techniques [2] in the mid-seventies. Although iterative methods lack the robustness of direct methods, they are effective for the large class of problems arising from partial differential equations of the elliptic type. An important gap in the literature concerns the development of truly general-purpose iterative solvers that could replace direct methods with a minimum risk of failure. A comparison between existing software based on direct methods and software based on iterative methods reveals that the direct solvers have evolved quite differently and have acquired a level of sophistication that far exceeds that of iterative methods.

In order to be able to enhance robustness of iterative solvers, we should be able to determine, e.g., by means of heuristics, whether or not a given preconditioner is suitable for the problem at hand. If not, one can attempt another possible iterative method/preconditioner and switch periodically if necessary. It is desirable to be able to switch within the outer iteration instead of restarting. For the GMRES algorithm [5], this can be accomplished with the help of a rather simple modification of the standard algorithm, referred to as the flexible GMRES (FGMRES), which is presented in this paper. An important property of FGMRES is that it satisfies the residual norm minimization property over the preconditioned Krylov subspace just as in the standard GMRES algorithm [5].

For motivation, we mention that there are cases in which relaxation-type preconditioners are more attractive than the usual ILU preconditioners. These include the case in which a red-black ordering is used. In this situation, the single-step SSOR preconditioner and the ILU preconditioning may perform very poorly. However, our experience is that if a higher level-of-fill ILU or a multiple-step SSOR (or SOR) preconditioner is used, then the preconditioned method can perform rather well [4]. In this situation, SOR and SSOR have a distinct advantage over the ILU-type preconditioners, in that they preserve their high degree of parallelism, which is of order N.

* Received by the editors November 6, 1991; accepted for publication (in revised form) April 3, 1992.

† Computer Science Department and Minnesota Supercomputer Institute, University of Minnesota, 200 Union Street S.E., Minneapolis, Minnesota 55455.

In contrast, a serious loss of parallelism is incurred for the incomplete factorization preconditioners with high level-of-fill. A few other advantages are discussed in [4]. Thus we wish to be able to apply an arbitrary number of SOR or SSOR steps in the preconditioning phase, for example, in order to solve the preconditioning system $My = v$ to a given tolerance. We may also wish to change the relaxation parameter ω, possibly at each GMRES step, in order to attempt to achieve optimality.

The FGMRES algorithm presented in this paper allows us to incorporate these changes in the preconditioner into the GMRES framework at little additional cost. To be precise, there is no additional cost incurred in the arithmetic but the memory requirement doubles. On the other hand, FGMRES may enable one to utilize the memory more efficiently since the vectors that are normally not being used in a given FGMRES step can be fully exploited to compute a preconditioned vector, e.g., via another GMRES run that uses these vectors. A few tests based on this approach will be presented in §3. We will present some applications of the technique to show how the method can be used to improve the robustness of the standard GMRES algorithm. We should point out that another illustration of the benefits of FGMRES in the finite element framework is described by Tezduyar, Behr, Abadi, and Ray [6].

2. Krylov subspaces with variable preconditioning. The basic principle of preconditionings is to use a Krylov subspace method for solving a modified system such as

$$AM^{-1}(Mx) = b.$$

Clearly, the matrix AM^{-1} need not be formed explicitly: we only need to solve $Mz = v$ whenever such an operation is required. Thus a fundamental requirement is that it should be easy to compute $M^{-1}v$ for an arbitrary vector v. In some cases, solving a linear system with the matrix M consists of forming an approximate solution by performing one or a few steps of a relaxation-type method, or a Chebyshev iteration. It is natural to consider preconditioners that do not use only a single step of an iterative method, but as many as are needed to solve a linear system within a given tolerance. In fact, this would be the equivalent of a higher level-of-fill in the usual ILU preconditioners, except that the preconditioner is no longer constant but is allowed to vary from one step to another in the outer iteration. A similar situation in which the preconditioner is "not constant" is when another Krylov subspace method, e.g., one that is based on the normal equations approach, is used as a preconditioner. These applications and others lead us to raise the question of whether or not it is possible to accommodate such *variations in the preconditioners* and still obtain an algorithm that satisfies an optimality property similar to the one satisfied by the original iterative method. This question has been avoided in the past because in the Hermitian case there does not seem to exist a version of the usual preconditioned conjugate gradient algorithm that satisfies a short vector recurrence and that allows the preconditioner to vary at each step. In the non-Hermitian case and for methods that do not rely on short vector recurrences, such as GMRES, variations in the preconditioner can be handled without difficulty, as we now show.

2.1. The algorithm. We start by describing the standard GMRES algorithm with right preconditioning and then show the flexible modification which allows such variations.

ALGORITHM 2.1. GMRES with right preconditioning.

```
1. Start: Choose $x_0$ and a dimension $m$ of the Krylov subspaces. Define an
   $(m+1) \times m$ matrix $\bar{H}_m$ and initialize all its entries $h_{i,j}$ to zero.
2. Arnoldi process:
   (a) Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
   (b) For $j = 1, \dots, m$ do
       • Compute $z_j := M^{-1}v_j$;
       • Compute $w := Az_j$;
       • For $i = 1, \dots, j$, do  $\left\{ \begin{array}{l} h_{i,j} := (w, v_i), \\ w := w - h_{i,j}v_i; \end{array} \right.$
       • Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.
   (c) Define $V_m := [v_1, \dots, v_m]$.
3. Form the approximate solution: Compute $x_m = x_0 + M^{-1}V_m y_m$ where
   $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \dots, 0]^T$.
4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.
```

The Arnoldi loop simply constructs an orthogonal basis of the preconditioned Krylov subspace

$$\text{Span}\{r_0, AM^{-1}r_0, \dots, (AM^{-1})^{m-1}r_0\}$$

by a modified Gram–Schmidt process, in which the new vector to be orthogonalized is obtained from the previous vector process. The last step in the above algorithm forms the solution as a linear combination of the preconditioned vectors $z_i = M^{-1}v_i, i = 1, \dots, m$. Because these vectors are all obtained by applying the same preconditioning matrix M^{-1} to the v's, we need not save them. We only need to apply M^{-1} to the linear combination of the $v's$, i.e., to $V_m y_m$. The question we can now ask is: what if we allowed the preconditioner to change at every step, i.e., z_j would be defined by

$$z_j = M_j^{-1} v_j,$$

but we saved these vectors to use them in updating x_m in step 3? In other words, we would like to consider the following "flexible" modification to the previous algorithm.

ALGORITHM 2.2. FGMRES: GMRES with variable preconditioning.

```
1. Start: Choose $x_0$ and a dimension $m$ of the Krylov subspaces. Define an
   $(m+1) \times m$ matrix $\bar{H}_m$ and initialize all its entries $h_{i,j}$ to zero.
2. Arnoldi process:
   (a) Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$.
   (b) For $j = 1, \dots, m$, do
       • Compute $z_j := M_j^{-1}v_j$;
       • Compute $w := Az_j$;
       • For $i = 1, \dots, j$, do  $\left\{ \begin{array}{l} h_{i,j} := (w, v_i), \\ w := w - h_{i,j}v_i; \end{array} \right.$
       • Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.
   (c) Define $Z_m := [z_1, \dots, z_m]$.
3. Form the approximate solution: Compute $x_m = x_0 + Z_m y_m$ where $y_m =$
   $\text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \dots, 0]^T$.
4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.
```

As can be observed, the only difference from the standard version is that we now save the preconditioned vectors z_i and update the solution using these vectors. It is clear that when $M_j = M$ for $j = 1, \ldots, m$, then the new method is mathematically equivalent to Algorithm 2.2. Note that we can define z_j in step 2(b) without reference to any preconditioner, i.e., we can simply pick a given new vector z_j. We would like to mention that the technique presented above can be viewed as an extension of a strategy presented in [1] in the context of using Krylov subspace methods for solving nonlinear equations. More recently, van der Vorst and Vuik developed a family of algorithms that have the same feature as FGMRES in that they also allow variations in the preconditioner [7].

2.2. Some basic properties. One notable difference between FGMRES and the usual GMRES algorithm is that the action of AM_j^{-1} on a vector v of the Krylov subspace is no longer in the span of V_{m+1}. Instead, it is easy to show that the following equality takes place:

$$AZ_m = V_{m+1}\bar{H}_m \ . \tag{1}$$

This replaces the simpler relation

$$(AM^{-1})V_m = V_{m+1}\ \bar{H}_m,$$

which holds for the standard preconditioned GMRES [5]. Following [5] we will denote by H_m the $m \times m$ matrix obtained from $\bar{H}_m$ by deleting its last row and by $\hat{v}_{j+1}$ the vector w obtained at the end of step 2(b) of the algorithm, i.e., the vector obtained before normalizing w to get v_{j+1}. Then an alternative to (1) that is valid even when $h_{m+1,m} = 0$ is the following:

$$AZ_m = V_m H_m + \hat{v}_{m+1} e_m^T \ . \tag{2}$$

We will now prove an optimality property similar to the one that defines GMRES. Consider the residual vector for an arbitrary vector $z = x_0 + Z_m y$ in the affine space $x_0 + \text{span}\{Z_m\}$. We have

$$\begin{aligned} b - Az &= b - A(x_0 + Z_m y) \\ &= r_0 - AZ_m y \qquad (3) \\ &= \beta v_1 - V_{m+1}\bar{H}_m y \\ &= V_{m+1}[\beta e_1 - \bar{H}_m y]. \qquad (4) \end{aligned}$$

If we denote by $J_m(y)$ the function

$$J_m(y) = \|b - A[x_0 + Z_m y]\|_2,$$

we observe that by (4) and the fact that V_{m+1} is unitary,

$$J_m(y) = \|\beta e_1 - \bar{H}_m y\|_2. \tag{5}$$

Since step 3 of Algorithm 2.2 minimizes this norm over all vectors y in R^m to yield y_m, it is clear that the approximate solution $x_m = x_0 + Z_m y_m$ has the smallest residual norm in $x_0 + \text{Span}\{Z_m\}$. Thus we have proved the following result.

PROPOSITION 2.1. *The approximate solution x_m obtained at step m minimizes the residual norm $\|b - Ax_m\|_2$ over $x_0 + \text{Span}\{Z_m\}$.*

We will now examine a case of breakdown in FGMRES, which occurs when $h_{j+1,j} = 0$ in the last part of step 3 in Algorithm 2.2. In this situation, the vector v_{j+1} cannot be computed and the algorithm breaks down. For the standard GMRES algorithm this is not a problem because when this breakdown occurs, then the approximate solution x_j is exact. In fact, breakdown is equivalent to convergence. In FGMRES this is no longer true. More specifically, we have the following result.

PROPOSITION 2.2. *Assume that $\beta = \|r_0\|_2 \neq 0$ and that $j-1$ steps of* FGMRES *have been successfully performed, i.e., that $h_{i+1,i} \neq 0$ for $i < j$. In addition, assume that the matrix H_j is nonsingular. Then x_j is exact if and only if $h_{j+1,j} = 0$.*

Proof. If $h_{j+1,j} = 0$, then we have the relation $AZ_j = H_j V_j$, and as a result,

$$J_j(y) = \|\beta v_1 - AZ_j y_j\|_2 = \|\beta v_1 - V_j H_j y_j\|_2 = \|\beta e_1 - H_j y_j\|_2.$$

If we assume that H_j is nonsingular, then the above function is minimized for $y_j = H_j^{-1}(\beta e_1)$ and the corresponding minimum norm reached is zero, i.e., x_j is exact.

Conversely, if x_j is exact, then from (2) and (3) we have

$$0 = b - Ax_j = V_j[\beta_1 e_1 - H_j y_j] + \hat{v}_{j+1} e_j^T y_j. \tag{6}$$

If the last component of y_j is zero, then (6) would mean that $H_j y_j = \beta e_1$ and, since $h_{i+1,i} \neq 0$ for $1 \leq i \leq j-1$, a simple back-substitution starting from the last equation will show that all components of y_j are zero. This would imply that $\beta = 0$ and contradict the assumption. Hence $e_j^T y_j \neq 0$. Therefore, since $\hat{v}_{j+1}$ is orthogonal to $v_1, \ldots, v_j$ the only way in which (6) can hold is that $\beta_1 e_1 - H_j y_j = 0$ and $\hat{v}_{j+1} = 0$, which implies $h_{j+1,j} = 0$. □

Note that the only difference between this result and the one in [5] concerning the standard GMRES algorithm is that we must make the additional assumption that H_j is nonsingular since this is no longer implied by the nonsingularity of A.

A consequence of the result is that if at a given step j, we have $Az_j = v_j$, i.e., if the preconditioning is "exact" at step j, then the approximation x_j will be exact if in addition H_j is nonsingular. This is because $w = Az_j$ is linearly dependent on previous v_i's (it is equal to v_j), and as a result we will obtain $\hat{v}_{j+1} = 0$ after the orthogonalization process.

A difficulty with the theory of the new algorithm is that we cannot prove general convergence results such as those in [5]. This is because the subspace of approximants is no longer a standard Krylov subspace and we have no isomorphism with the space of polynomials. However, the optimality property of Proposition 2.1 can be exploited in some specific situations. For example, if within each outer iteration we select *at least one* of the vectors z_j to be a steepest descent direction vector, e.g., for the function $F(x) = \|b - Ax\|_2^2$, then FGMRES is guaranteed to converge, independently of m.

2.3. Practical considerations and applications. The additional cost incurred by the flexible variant over the standard algorithm is only in the extra memory required to save the set of vectors $\{z_j\}_{j=1,\ldots,m}$. On the other hand, the added advantage of *flexibility* may certainly be worth this extra cost. There are a few applications in which this flexibility can be quite helpful, especially in the context of developing robust iterative methods or for developing preconditioners for massively parallel computers. Here is a sample of possible applications.

1. Use of *any* iterative techniques as preconditioners: block-SOR, SSOR, ADI, multigrid, etc., but also GMRES, CGNR, CGS, etc.

2. Use of chaotic relaxation-type preconditioners (e.g., in a parallel computing environment).

3. Mixing preconditioners to solve a given problem.

For an example of (3), see the recent work by Tezduyar, Behr, Adadi, and Ray [6], in which two types of preconditioners are alternately applied at each FGMRES step to mix the effects of "local" and "global" interactions. Tezduyar et al. reported good performance with this procedure, much better than using either of the two preconditioners by itself.

Note that any iterative method can now be used as a preconditioner. For example, we will show how to make some nonnegligible improvements to the performance of the basic GMRES algorithm by using GMRES as preconditioner to itself (using for memory space the unused vectors at the ith step of FGMRES(m)).

Preconditioners of particular interest within this framework are relaxation-type techniques. As an example, the SSOR preconditioning matrix defined by

$$M_{\text{SSOR}}(A) = (D - \omega E)D^{-1}(D - \omega F),$$

in which $-E$ is the strict lower part of A, $-F$ is the strict upper part of A, and D is the diagonal of A, has some important advantages, some of which have been briefly discussed in the introduction. In the context of preconditioning, it is customary to just take $\omega = 1$, as the gains from selecting an optimal ω are typically small. However, it is clear that one can use different values of ω at each step of FGMRES and this can open up the possibility of using heuristics to determine the best ω dynamically, by simply monitoring convergence. Alternatively, a mixture of ω's can be initially selected and then used cyclically, instead of arbitrarily taking only $\omega = 1$ as is usually done. We would also like to make an important point concerning SOR as a preconditioner. The usual one-step SOR is not popular as a preconditioner, mainly because it tends to distribute the eigenvalues of the preconditioned matrix around a circle (e.g., for the model problem). This is not very desirable for a CG-type solver. However, we found that when using multiple steps, SOR is often more economical than SSOR as a preconditioner.

We will not discuss these applications here but refer the reader to [4]. Rather, we would like to demonstrate the flexibility of FGMRES by simply combining it with other iterative methods to improve its robustness. We are particularly interested in combinations with CGNR, and with the standard GMRES itself. The reason why we chose CGNR is that we know that the method is globally convergent and as a result, adding one direction vector to the standard Krylov subspace will guarantee global convergence because of the optimality of FGMRES. We should add, however, that guaranteeing global convergence is not the ultimate goal, since the convergence can still be too slow to be of any practical value.

3. Numerical experiments.

For test purposes we consider the problems arising from the centered difference discretization of problems of the form

$$-\Delta u + \gamma(xu_x + yu_y) + \beta u = f \tag{7}$$

on square regions with zero Dirichlet boundary conditions. In our first test we select the parameters $\gamma = 10$ and $\beta = -100$ to make the problem indefinite. The grid consists of a square of 32 internal mesh points in each direction leading to a matrix of size $N = 1024$. The right-hand side is selected once the matrix is constructed so that the solution is known to be $x = (1, 1, \dots, 1)^T$. In all methods the initial vector is chosen so that its ith component for $i = 1, \dots, n$ is defined by $x_0(i) = i$. We have compared the following methods.

1. A standard ILU(0) preconditioned GMRES(m) iteration with $m = 20$ direction vectors.

2. The CGNR iteration (conjugate gradient, normal equations) using an ILQ preconditioner with level-of-fill equal to 5; see [3] for details on these preconditioners. The idea of ILQ is to perform an incomplete Gram–Schmidt factorization on the rows of A. The normal equations are preconditioned using this factorization and the conjugate gradient method (CGNR version) is used to solve these equations.

3. The FGMRES iteration using (the unpreconditioned) GMRES itself as a preconditioner. As was briefly mentioned in the introduction, this is run as follows. We observe that at step i of the FGMRES iteration, the space for the vectors $v_{i+2}, \ldots, v_{m+1}$ and $z_k, k = i+1, \ldots, m$ is unused. However, they can be exploited to generate a preconditioned version of v_i by running a standard GMRES iteration using $2m - i - 1$ direction vectors. In fact, the vectors v_j and z_j occupy the same array $wk(1:n, 1:2m)$ and the z_k's are stored backwards starting in column $2m$ to avoid collisions. Note that for fairness in the comparisons, we use here $m = 10$ so that the total number of vectors needed is $2m = 20$, the same as is required for the ILU(0)-GMRES in 1.

4. The FGMRES iteration using ILU(0)-GMRES as a preconditioner. This is similar to the previous method except that the method used to precondition is ILU(0)-GMRES instead of the unpreconditioned GMRES.

5. The FGMRES iteration using k steps of the unpreconditioned CGNR iteration as a preconditioner. In our first test we took $k = 5$ and in the second, $k = 2$. Other values of k have also been tested and performed similarly.

Figure 1 is a plot of the residual norm achieved by these five methods against the number of operations (in millions) required to reach that level for the first test problem. We chose to plot the residuals versus the number of operations because the number of iterations is no longer significant for making comparisons since the cost of each iteration can be quite different for each of the methods compared. Observe that the FGMRES iteration using the unpreconditioned GMRES (method 3) converges, although slowly, whereas the standard GMRES(20) (not shown) as well as the ILU(0) preconditioned GMRES(20) both stagnate. More interesting is the convergence of FGMRES with ILU(0)-GMRES as a preconditioner. ILU(0)-GMRES(20) alone fails to converge (stopped after 700 steps). Used as a preconditioner, the technique converges in 15 outer iterations and yields the second best performance in this test. In this test CGNRILQ(5) performed quite well.

In our second test we took the same partial differential equation as before, but with the parameters $\gamma = 1000$ and $\beta = 10.0$, to make the problem highly nonsymmetric. The grid is the same as before and so the size of the matrix is still 1024. The right-hand side and the starting vector are generated similarly to the previous example. We have run the same methods as before, except that the number of CGNR steps used in FGMRES-CGNR is two instead of five. The results are shown in Fig. 2. Note that this time, CGNR-ILQ does not perform as well. In fact, some of the conclusions of the previous test are reversed. Thus, FGMRES-ILU(0)-GMRES is now outperformed by the simpler ILU(0)-GMRES(20). The FGMRES with unpreconditioned GMRES outperforms the FGMRES with ILU(0)-preconditioned GMRES by a slight margin. In addition, FGMRES with CGNR preconditioner using just two steps of CGNR is now the overall best.

4. Conclusion.

An interesting observation from the above experiments is that for indefinite and/or highly nonsymmetric matrices the performance of a given precon-

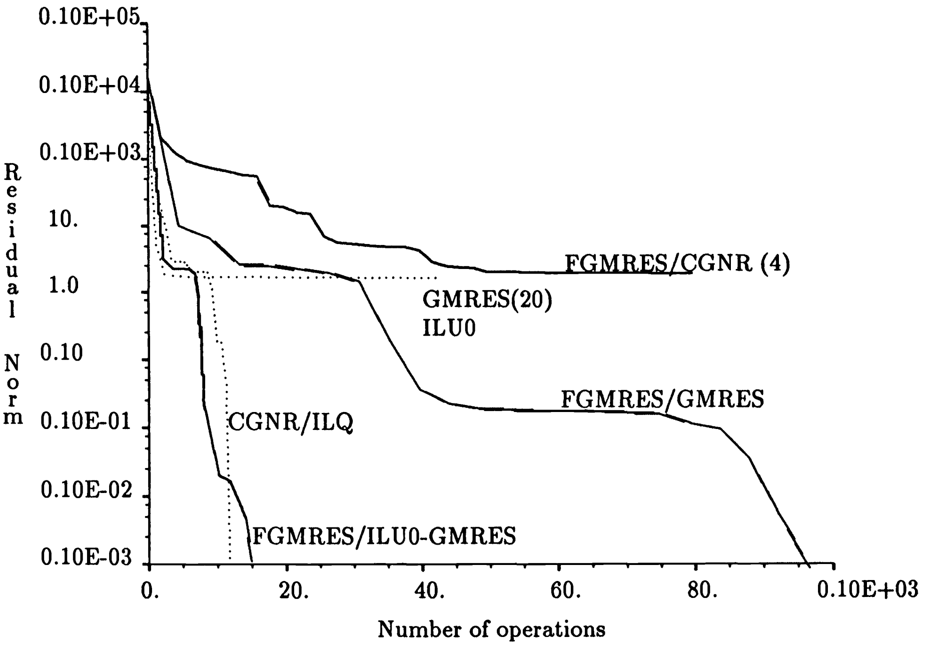


FIG. 1. *Performance of different iterative methods for first test problem.*

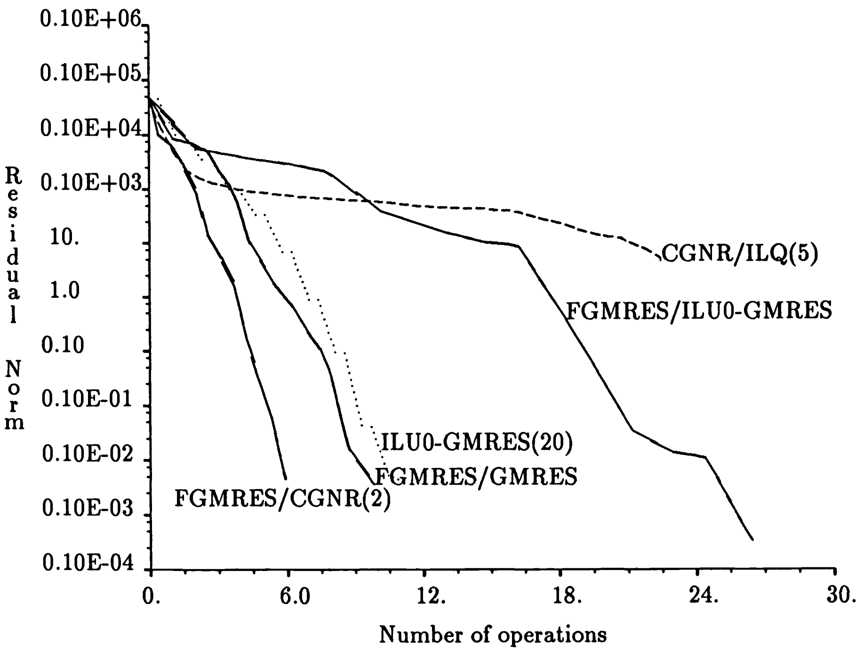


FIG. 2. *Performance of different iterative methods for second test problem.*

ditioner can be unpredictable. In these situations, it is essential to be able to switch preconditioners in order to improve robustness. FGMRES is an algorithm that allows arbitrary changes in the preconditioner and can be used to this end. There are many other uses of the flexible variant of GMRES. In addition, the difference in the coding of the two methods is so small that they can both be implemented, with no loss of efficiency, in a single subroutine that incorporates an option parameter.

Acknowledgments. The author would like to acknowledge the support of the Minnesota Supercomputer Institute, which provided the computer facilities and other resources for this research. The author is indebted to Henk van der Vorst for pointing out an error in an earlier version of this paper.

REFERENCES

[1] P. N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 450–481.

[2] 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.

[3] Y. Saad, *Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals*, SIAM J. Numer. Anal., 20 (1983), pp. 784–811.

[4] ———, *Highly parallel preconditioners for general sparse matrices*, Tech. Rep. 92-087, Army High Performance Computing Research Center, Minneapolis, MN, 1992.

[5] 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.

[6] T. E. Tezduyar, M. Behr, S. K. A. Abadi, and S. E. Ray, *A mixed* CEBE/CC *preconditioning for finite element computations*, Tech. Rep. UMSI 91/160, Univ. of Minnesota, Minnesota Supercomputing Institute, Minneapolis, MN, June 1991.

[7] H. A. van der Vorst and C. Vuik, GMRESR: *A family of nested GMRES methods*, Tech. Rep. 91-80, Delft University of Technology, Mathematics and Informatics, Delft, the Netherlands, 1991.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 470–482, March 1993

A TRANSPOSE-FREE QUASI-MINIMAL RESIDUAL ALGORITHM FOR NON-HERMITIAN LINEAR SYSTEMS*

ROLAND W. FREUND†

Abstract. The biconjugate gradient method (BCG) for solving general non-Hermitian linear systems $Ax = b$ and its transpose-free variant, the conjugate gradients squared algorithm (CGS), both typically exhibit a rather irregular convergence behavior with wild oscillations in the residual norm. Recently, Freund and Nachtigal proposed a BCG-like approach, the quasi-minimal residual method (QMR), that remedies this problem for BCG and produces smooth convergence curves. However, like BCG, QMR requires matrix-vector multiplications with both the coefficient matrix A and its transpose A^T. In this note, it is demonstrated that the quasi-minimal residual approach can also be used to obtain a smoothly convergent CGS-like algorithm that does not involve matrix-vector multiplications with A^T. It is shown that the resulting transpose-free QMR method (TFQMR) can be implemented very easily by changing only a few lines in the standard CGS algorithm. Finally, numerical experiments are reported.

Key words. non-Hermitian linear systems, biconjugate gradients, transpose-free, conjugate gradients squared, quasi-minimal residual property

AMS(MOS) subject classification. 65F10

1. Introduction.

The classical conjugate gradient method (CG) [11] is one of the most powerful iterative schemes for solving large sparse linear systems

$$Ax = b \tag{1.1}$$

with Hermitian positive definite coefficient matrices A. The biconjugate gradient algorithm (BCG) [13], [3] is the "natural" extension of CG to linear systems (1.1) with general non-Hermitian nonsingular coefficient matrices. However, unlike CG, the BCG iterates are not characterized by a minimization property, which means that the algorithm can exhibit—and typically does—a rather irregular convergence behavior with wild oscillations in the residual norm. Furthermore, in the BCG algorithm, even breakdowns—more precisely, division by 0—may occur.

Recently, Freund and Nachtigal [7] proposed a BCG-like approach, the quasi-minimal residual method (QMR), that remedies the problems of BCG. The QMR iterates are defined by a quasi minimization of the residual norm, which leads to smooth convergence curves. Furthermore, QMR can be implemented based on a look-ahead version [14], [6] of the nonsymmetric Lanczos algorithm [12], which avoids possible breakdowns of the process, except for the very special situation of an incurable breakdown.

Except for special cases [8], such as complex symmetric matrices [4], BCG and QMR require matrix-vector multiplications with both the coefficient matrix A of (1.1) and its transpose A^T. This is a disadvantage for certain applications, such as linear systems arising in ordinary differential equation solvers [9], where A^T is not readily available. Sonneveld [15] derived a variant of BCG, the conjugate gradients squared algorithm (CGS), that does not involve A^T. However, like BCG, CGS also

* Received by the editors September 9, 1991; accepted for publication (in revised form) April 3, 1992. This research was supported by the Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration and the Universities Space Research Association.

† Research Institute for Advanced Computer Science, Mail Stop T041-5, NASA Ames Research Center, Moffett Field, California 94035. Present address, AT&T Bell Laboratories, Room 2C-420, 600 Mountain Avenue, Murray Hill, New Jersey 07974-0636.

exhibits rather erratic convergence behavior. Van der Vorst [16] proposed the Bi-CGSTAB algorithm, which uses local steepest descent steps to obtain a more smoothly convergent CGS-like process. While Bi-CGSTAB seems to work well in many cases, it still exhibits irregular convergence behavior for some difficult problems. Also, Bi-CGSTAB can converge considerably slower than CGS.

In this note, we demonstrate that the quasi-minimal residual approach can also be used to obtain a smoothly convergent CGS-like algorithm that does not require matrix-vector multiplications with A^T. We show that the resulting transpose-free QMR method (TFQMR) can be implemented very easily, by changing only a few lines in the standard CGS algorithm. We stress that the proposed TFQMR method and the original QMR algorithm [7] are not mathematically equivalent. In particular, the sets of iterates generated by the two schemes are different.

The rest of this paper is organized as follows. In §2, we briefly review the CGS algorithm. In §3, we present the transpose-free quasi-minimal residual approach. To derive an actual implementation, we first present an auxiliary result in §4 and then present the resulting TFQMR algorithm in §5. Numerical examples are reported in §6, and in §7, we make some concluding remarks.

Throughout the paper, all vectors and matrices are allowed to have real or complex entries. As usual, M^T and M^H denote the transpose and conjugate transpose of M, respectively. The vector norm $\|x\| = \sqrt{x^H x}$ is always the Euclidean norm, and $\|M\| = \max_{\|x\|=1} \|Mx\|$ is the corresponding matrix norm. The notation

$$K_n(c, M) := \operatorname{span}\{c, Mc, \ldots, M^{n-1}c\}$$

is used for the nth Krylov subspace of $\mathbb{C}^N$ generated by $c \in \mathbb{C}^N$ and the $N \times N$ matrix M. For $q \in \mathbb{R}$, $\lfloor q \rfloor$ is the largest integer $\leq q$. The set of all complex polynomials of degree at most n is denoted by

$$\mathcal{P}_n := \{\varphi(\lambda) \equiv \gamma_0 + \gamma_1\lambda + \cdots + \gamma_n\lambda^n \mid \gamma_0, \gamma_1, \ldots, \gamma_n \in \mathbb{C}\}.$$

The coefficient matrix A of (1.1) is always assumed to be a nonsingular, in general non-Hermitian, $N \times N$ matrix, and $b \in \mathbb{C}^N$. Generally, $x_n \in \mathbb{C}^N$, $n = 0, 1, \ldots$, denote iterates for (1.1) with corresponding residual vectors $r_n := b - Ax_n$. If necessary, quantities of different algorithms will be distinguished by superscripts, e.g., x_n^{BCG} and x_n^{CGS}.

2. The CGS algorithm.

Let $x_0 \in \mathbb{C}^N$ be any initial guess for (1.1) with residual vector $r_0 = b - Ax_0$, and let $\tilde{r}_0 \in \mathbb{C}^N$ be any vector such that $\tilde{r}_0^H r_0 \neq 0$, e.g., $\tilde{r}_0 = r_0$. The nth iterate, x_n^{BCG}, generated by the BCG process is defined by the Galerkin-type condition

$$w^H(b - Ax_n^{\mathrm{BCG}}) = 0 \quad \text{for all } w \in K_n(\tilde{r}_0, A^H), \quad x_n^{\mathrm{BCG}} \in x_0 + K_n(r_0, A). \tag{2.1}$$

Clearly, the corresponding residual vector r_n^{BCG} is of the form

$$r_n^{\mathrm{BCG}} = \varphi_n(A) r_0, \quad \text{where} \quad \varphi_n \in \mathcal{P}_n \quad \text{and} \quad \varphi_n(0) = 1. \tag{2.2}$$

Sonneveld [15] observed that the iterates $x_n \in x_0 + K_{2n}(r_0, A)$ whose residual vectors are just the "squares" of (2.2), i.e.,

$$r_n = (\varphi_n(A))^2 r_0, \tag{2.3}$$

can be computed by means of a BCG-like process that does not involve A^H. The resulting process is as follows.

ALGORITHM 2.1 (CGS ALGORITHM [15]).

```
(1) Start:
    (a) Choose $x_0 \in \mathbb{C}^N$;
    (b) Set $p_0 = u_0 = r_0 = b - Ax_0$, $v_0 = Ap_0$;
    (c) Choose $\tilde{r}_0$ such that $\rho_0 = \tilde{r}_0^H r_0 \neq 0$.
(2) For $n = 1, 2, \ldots$, do:
    (a) Set $\sigma_{n-1} = \tilde{r}_0^H v_{n-1}$, $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$;
            $q_n = u_{n-1} - \alpha_{n-1} v_{n-1}$;
    (b) Set $x_n = x_{n-1} + \alpha_{n-1}(u_{n-1} + q_n)$;
            $r_n = r_{n-1} - \alpha_{n-1} A(u_{n-1} + q_n)$;
        If $x_n$ has converged: stop;
    (c) Set $\rho_n = \tilde{r}_0^H r_n$, $\beta_n = \rho_n/\rho_{n-1}$;
            $u_n = r_n + \beta_n q_n$;
            $p_n = v_n + \beta_n(q_n + \beta_n p_{n-1})$;
            $v_n = Ap_n$.
```

In exact arithmetic, Algorithm 2.1 terminates after a finite number, say n_*, of iterations. Usually, $x_{n_*} = A^{-1}b$ is the solution of (1.1). However, for general A, the possibility cannot be excluded that Algorithm 2.1 terminates prematurely with $\sigma_{n_*} = 0$ or $\rho_{n_*} = 0$, before the solution of (1.1) has been found. If this happens, a continuation of the process would lead to division by 0 in step (2a) or (2c) of the next iteration, and therefore premature termination is also referred to as a "breakdown" of the algorithm. Fortunately, breakdowns are very rare in practice and, furthermore, they can be overcome by using look-ahead strategies (cf. §7). Here, for simplicity, we will consider only the standard CGS Algorithm 2.1, without look-ahead. We note that a modified CGS algorithm that avoids exact breakdowns was recently given by Brezinski and Sadok [1].

In the sequel, we always assume that $n \in \{1, 2, \ldots, n_*\}$. Note that

$$\alpha_{n-1} \neq 0 \quad \text{for all } n. \tag{2.4}$$

Moreover, we will make use of the relations

$$u_{n-1} = \varphi_{n-1}(A)\psi_{n-1}(A)r_0 \quad \text{and} \quad q_n = \varphi_n(A)\psi_{n-1}(A)r_0, \tag{2.5}$$

which are derived in [15]. Here the φ_n's are given by (2.2), and the polynomials ψ_n can be updated by means of

$$\psi_n(\lambda) \equiv \varphi_n(\lambda) + \beta_n \psi_{n-1}(\lambda), \tag{2.6}$$

where $\psi_0(\lambda) \equiv 1$. Finally, we note that

$$\varphi_n(\lambda) \equiv \varphi_{n-1}(\lambda) - \alpha_{n-1}\lambda\psi_{n-1}(\lambda). \tag{2.7}$$

3. The quasi-minimal residual approach. By (2.2) and (2.3), the CGS iterates are defined implicitly via the Galerkin condition (2.1), but they do not satisfy a residual norm minimization property. In this section, we demonstrate that, using the same sequence of vectors u_{n-1} and q_n as generated by Algorithm 2.1, one can also define iterates with a quasi-minimization property.

We set

$$(3.1)\qquad y_m = \begin{cases} u_{n-1} & \text{if } m = 2n-1 \text{ is odd,} \\ q_n & \text{if } m = 2n \text{ is even,} \end{cases}$$

and

$$(3.2)\qquad w_m = \begin{cases} (\varphi_n(A))^2 r_0 & \text{if } m = 2n+1 \text{ is odd,} \\ \varphi_n(A)\varphi_{n-1}(A) r_0 & \text{if } m = 2n \text{ is even.} \end{cases}$$

Note that, by (2.3),

$$(3.3)\qquad w_{2n+1} = r_n^{\text{CGS}}, \qquad n = 1, 2, \ldots, n_*.$$

In the sequel, it is always assumed that $m \in \{1, 2, \ldots, 2n_*\}$. Using (2.5) and (2.7), one readily verifies that the vectors (3.1) and (3.2) are related by

$$(3.4)\qquad Ay_m = \frac{1}{\alpha_{\lfloor (m-1)/2 \rfloor}} (w_m - w_{m+1}).$$

Note that, by (2.4), the denominator in (3.4) is guaranteed to be different from 0. Setting

$$\begin{aligned} Y_m &= [y_1 \quad y_2 \quad \cdots \quad y_m], \\ W_{m+1} &= [w_1 \quad w_2 \quad \cdots \quad w_m \quad w_{m+1}], \end{aligned}$$

we can rewrite the relations (3.4) in matrix form

$$(3.5)\qquad AY_m = W_{m+1} B_m^{(e)},$$

where

$$(3.6)\qquad B_m^{(e)} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 1 \\ 0 & \cdots & 0 & -1 \end{bmatrix} \cdot \left(\operatorname{diag}(\alpha_0, \alpha_0, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor})\right)^{-1}$$

is an $(m+1) \times m$ lower bidiagonal matrix. Finally, note that in view of (2.6), (2.7), and (2.4), the polynomials φ_n and ψ_n are both of full degree n, and with (2.5) and (3.1), it follows that

$$(3.7)\qquad K_m(r_0, A) = \operatorname{span}\{y_1, y_2, \ldots, y_m\} = \{Y_m z \mid z \in \mathbb{C}^m\}.$$

After these preliminaries, we now begin our derivation of the quasi-minimal residual approach. By (3.7), any possible iterate $x_m \in x_0 + K_m(r_0, A)$ can be written in the form

$$(3.8)\qquad x_m = x_0 + Y_m z \quad \text{for some } z \in \mathbb{C}^m.$$

By (3.5) and since $w_1 = r_0$ (see (3.3)), the corresponding residual vector satisfies

$$r_m = r_0 - AY_m z = W_{m+1}\left(e_1^{(m+1)} - B_m^{(e)} z\right), \tag{3.9}$$

where $e_1^{(m+1)} = [1 \quad 0 \quad \cdots \quad 0]^T \in \mathbb{R}^{(m+1)}$. Ideally, we would like to choose the free parameter vector z in (3.8) such that the norm $\|r_m\|$ of (3.9) is minimized. However, since, in general, the matrix W_{m+1} is dense and its columns are not orthogonal to each other, this would require the solution of a dense $N \times (m+1)$ least-squares problem. Instead, we minimize only the length of the coefficient vector in the representation (3.9) of r_m. More precisely, let

$$\Omega_{m+1} = \operatorname{diag}(\omega_1, \omega_2, \ldots, \omega_{m+1}), \quad \omega_k > 0, \quad k = 1, 2, \ldots, m+1, \tag{3.10}$$

be any scaling matrix, and rewrite (3.9) as

$$r_m = W_{m+1}\Omega_{m+1}^{-1}\left(f_{m+1} - H_m^{(e)} z\right), \tag{3.11}$$

where

$$f_{m+1} = \omega_1 e_1^{(m+1)} \quad \text{and} \quad H_m^{(e)} = \Omega_{m+1} B_m^{(e)}. \tag{3.12}$$

We then define the mth iterate x_m of TFQMR by

$$x_m = x_0 + Y_m z_m, \tag{3.13}$$

where $z_m \in \mathbb{C}^m$ is the solution of the least-squares problem

$$\tau_m := \left\|f_{m+1} - H_m^{(e)} z_m\right\| = \min_{z \in \mathbb{C}^m} \left\|f_{m+1} - H_m^{(e)} z\right\|. \tag{3.14}$$

Note that by (3.6), (3.10), and (3.12), the matrix $H_m^{(e)}$ has full column rank m, and thus z_m is uniquely defined by (3.14). Furthermore, as we will show in the next two sections, the solution of (3.14) and hence x_m can be computed by means of simple recurrences.

Clearly, the iterates x_m still depend on the choice of the weights ω_k in (3.10). The standard strategy is to set

$$\omega_k = \|w_k\|, \qquad k = 1, 2, \ldots, m+1, \tag{3.15}$$

which means that all columns of $W_{m+1}\Omega_{m+1}^{-1}$ in the representation (3.11) are treated equally and scaled to be unit vectors. However, we will also consider a slightly less expensive choice of the weights in §5.

For the derivation of an actual implementation of the TFQMR method, the auxiliary iterates

$$\tilde{x}_m = x_0 + Y_m \tilde{z}_m, \quad \text{where} \quad \tilde{z}_m = H_m^{-1} f_m, \tag{3.16}$$

will be used. Here H_m denotes the $m \times m$ matrix obtained by deleting the last row of $H_m^{(e)}$. Using (3.6), (3.10), and (3.12), one readily verifies that H_m is indeed nonsingular and that

$$\tilde{z}_m = [\alpha_0 \quad \alpha_0 \quad \alpha_1 \quad \cdots \quad \alpha_{\lfloor (m-1)/2 \rfloor}]^T, \tag{3.17}$$

$$\omega_{m+1} = \left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\|. \tag{3.18}$$

Finally, by comparing (3.16) and (3.17) with the update formula for the iterates x_n^{CGS} in step (2b) of Algorithm 2.1, we conclude that

$$\tilde{x}_{2n} = x_n^{\mathrm{CGS}}. \tag{3.19}$$

4. A lemma.

In this section, we show that the solutions of least-squares problems of the type (3.14) can be updated easily from step to step by using the auxiliary quantity $\tilde{z}_m$ defined in (3.16). This is true for more general upper Hessenberg matrices $H_m^{(e)}$, not only for the particular matrices given by (3.12) and (3.6). Therefore, we prove the following more general result, which might also be useful in other situations.

LEMMA 4.1. *Let $\omega_1 > 0$, $m \geq 1$, and*

$$H_m^{(e)} = \begin{bmatrix} H_m \\ h_{m+1,m} e_m^T \end{bmatrix} = \begin{bmatrix} H_{m-1}^{(e)} & * \\ 0 & h_{m+1,m} \end{bmatrix}, \tag{4.1}$$

where $e_m^T = [0 \ \cdots \ 0 \ 1]$, be an $(m+1) \times m$ upper Hessenberg matrix of full column rank m. For $k = m-1,\ m$, let $z_k \in \mathbb{C}^k$ denote the solution of the least-squares problem

$$\tau_k := \min_{z \in \mathbb{C}^k} \left\| f_{k+1} - H_k^{(e)} z \right\|, \quad \textit{where } f_{k+1} = \omega_1 e_1^{(k+1)} \in \mathbb{R}^{k+1}. \tag{4.2}$$

Moreover, assume that the $m \times m$ matrix H_m in (4.1) is nonsingular, and set $\tilde{z}_m := H_m^{-1} f_m$. Then

$$z_m = (1 - c_m^2) \begin{bmatrix} z_{m-1} \\ 0 \end{bmatrix} + c_m^2 \tilde{z}_m, \tag{4.3}$$

$$\tau_m = \tau_{m-1} \vartheta_m c_m, \tag{4.4}$$

where

$$\vartheta_m = \frac{1}{\tau_{m-1}} \left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\|, \qquad c_m = \frac{1}{\sqrt{1 + \vartheta_m^2}}. \tag{4.5}$$

Proof. Let $k = m-1$ or $k = m$. The standard approach (see, e.g., [10, Chap. 6]) for solving the least-squares problem (4.2) is based on a QR factorization of $H_k^{(e)}$,

$$Q_{k+1} H_k^{(e)} = \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \tag{4.6}$$

where Q_{k+1} is unitary and R_k is a nonsingular upper triangular matrix. The solution of (4.2) is then given by

$$z_k = R_k^{-1} g_k, \quad \text{where } \begin{bmatrix} g_k \\ \tilde{\gamma}_{k+1} \end{bmatrix} = Q_{k+1} f_{k+1}, \quad \tilde{\gamma}_{k+1} \in \mathbb{C}, \tag{4.7}$$

and

$$\tau_k = |\tilde{\gamma}_{k+1}|. \tag{4.8}$$

Furthermore, since $H_k^{(e)}$ is an upper Hessenberg matrix, we can choose the matrix Q_{k+1} as a product of k Givens rotations. In particular, the factorization (4.6) for $k = m$ can then be easily updated from the one for $k = m - 1$, by setting

$$Q_{m+1} = \begin{bmatrix} I_{m-1} & 0 & 0 \\ 0 & c_m & -\overline{s_m} \\ 0 & s_m & c_m \end{bmatrix} \cdot \begin{bmatrix} Q_m & 0 \\ 0 & 1 \end{bmatrix}, \tag{4.9}$$

where $c_m \geq 0$ and $s_m \in \mathbb{C}$ are suitably chosen with $c_m^2 + |s_m|^2 = 1$. Also, note that

$$R_m = \begin{bmatrix} R_{m-1} & * \\ 0 & * \end{bmatrix}. \tag{4.10}$$

Using (4.9) and (4.1), we deduce from (4.6) (with $k = m$) that

$$Q_m H_m = \operatorname{diag}(1, \ldots, 1, c_m) R_m, \tag{4.11}$$

which, in particular, implies $c_m > 0$. Moreover, with (4.9), one easily verifies that

$$g_m = \begin{bmatrix} g_{m-1} \\ \gamma_m \end{bmatrix}, \quad \gamma_m = c_m \tilde{\gamma}_m, \quad \text{and} \quad \tilde{\gamma}_{m+1} = s_m \tilde{\gamma}_m. \tag{4.12}$$

From (4.11), (4.12), and (4.7), it follows that

$$\tilde{z}_m = (Q_m H_m)^{-1} \begin{bmatrix} g_{m-1} \\ \tilde{\gamma}_m \end{bmatrix} = R_m^{-1} \begin{bmatrix} g_{m-1} \\ \gamma_m / c_m^2 \end{bmatrix} = R_m^{-1} \begin{bmatrix} R_{m-1} z_{m-1} \\ \gamma_m / c_m^2 \end{bmatrix}. \tag{4.13}$$

Using (4.7), (4.12), and (4.13), we obtain

$$z_m = (1 - c_m^2) R_m^{-1} \begin{bmatrix} R_{m-1} z_{m-1} \\ 0 \end{bmatrix} + c_m^2 \tilde{z}_m,$$

which, in view of (4.10), is just (4.3).

Since Q_{m+1} is unitary, the following holds:

$$\left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\| = \left\| Q_{m+1} f_{m+1} - Q_{m+1} H_m^{(e)} \tilde{z}_m \right\|,$$

and with (4.6), (4.7), (4.13), and (4.12), we arrive at

$$\begin{aligned} \left\| f_{m+1} - H_m^{(e)} \tilde{z}_m \right\| &= \left\| \begin{bmatrix} g_{m-1} \\ \gamma_m \\ \tilde{\gamma}_{m+1} \end{bmatrix} - \begin{bmatrix} g_{m-1} \\ \gamma_m / c_m^2 \\ 0 \end{bmatrix} \right\| \\ &= \left\| \begin{bmatrix} \gamma_m |s_m|^2 / c_m^2 \\ \tilde{\gamma}_{m+1} \end{bmatrix} \right\| = |\tilde{\gamma}_m| \frac{|s_m|}{c_m}. \end{aligned} \tag{4.14}$$

Finally, setting $\vartheta_m = |s_m|/c_m$ and using (4.8), the relations (4.5) and (4.4) follow from (4.14) and the identity on the right-hand side of (4.12). □

5. The TFQMR algorithm. We now apply Lemma 4.1 to the particular least-squares problem (3.14). From (4.3), it follows that the TFQMR iterates (3.13) and the auxiliary iterates (3.16) are connected by

$$x_m = (1 - c_m^2)x_{m-1} + c_m^2 \tilde{x}_m. \tag{5.1}$$

Moreover, by (4.5), (4.4), and (3.18), we have

$$\vartheta_m = \frac{\omega_{m+1}}{\tau_{m-1}}, \quad c_m = \frac{1}{\sqrt{1+\vartheta_m^2}}, \quad \text{and} \quad \tau_m = \tau_{m-1}\vartheta_m c_m. \tag{5.2}$$

Setting

$$d_m = \frac{1}{\alpha_{\lfloor (m-1)/2 \rfloor}} \left(\tilde{x}_m - x_{m-1}\right), \tag{5.3}$$

we can rewrite (5.1) in the form

$$x_m = x_{m-1} + \eta_m d_m, \quad \text{where} \quad \eta_m = c_m^2 \alpha_{\lfloor (m-1)/2 \rfloor}. \tag{5.4}$$

Next note that by (3.16) and (3.17), we have

$$\tilde{x}_m = \tilde{x}_{m-1} + \alpha_{\lfloor (m-1)/2 \rfloor} y_m,$$

and together with (5.3) and (5.4) (both with m replaced by $m-1$), it follows that

$$d_m = y_m + \frac{\vartheta_{m-1}^2 \eta_{m-1}}{\alpha_{\lfloor (m-1)/2 \rfloor}} d_{m-1}, \quad \text{where} \quad \vartheta_{m-1}^2 := \frac{1 - c_{m-1}^2}{c_{m-1}^2}. \tag{5.5}$$

Next we note that with (3.1) and (3.3), the recursions for q_n and u_n in step (2a) and (2c) of Algorithm 2.1 can be rewritten as follows:

$$y_{2n} = y_{2n-1} - \alpha_{n-1} v_{n-1} \quad \text{and} \quad y_{2n+1} = w_{2n+1} + \beta_n y_{2n}. \tag{5.6}$$

Also, by multiplying the update formula for p_n in step (2c) of Algorithm 2.1 by A, we obtain the recursion

$$v_n =: Ay_{2n+1} + \beta_n (Ay_{2n} + \beta_n v_{n-1}) \tag{5.7}$$

for $v_n = Ap_n$. By (3.4), the w_m's can be generated via

$$w_{m+1} = w_m - \alpha_{\lfloor (m-1)/2 \rfloor} Ay_m. \tag{5.8}$$

Finally, by combining the recurrences (5.2), (5.4)–(5.8), we obtain an actual implementation for computing the iterates of the TFQMR method. First, we state the resulting algorithm for the standard weighting strategy (3.15).

ALGORITHM 5.1 (TFQMR ALGORITHM WITH WEIGHTS (3.15)).

```
(1) Start:
    (a) Choose $x_0 \in \mathbb{C}^N$;
    (b) Set $w_1 = y_1 = r_0 = b - Ax_0$, $v_0 = Ay_1$, $d_0 = 0$;
            $\tau_0 = \|r_0\|$, $\vartheta_0 = 0$, $\eta_0 = 0$;
    (c) Choose $\tilde{r}_0$ such that $\rho_0 = \tilde{r}_0^H r_0 \neq 0$.
(2) For $n = 1, 2, \dots$, do:
```

```
(a) Set $\sigma_{n-1} = \tilde{r}_0^H v_{n-1}$, $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$;
    $y_{2n} = y_{2n-1} - \alpha_{n-1} v_{n-1}$;
(b) For $m = 2n-1, 2n$ do:
    • Set $w_{m+1} = w_m - \alpha_{n-1} A y_m$;
    •     $\vartheta_m = \|w_{m+1}\|/\tau_{m-1}$, $c_m = 1/\sqrt{1+\vartheta_m^2}$;
    •     $\tau_m = \tau_{m-1}\vartheta_m c_m$, $\eta_m = c_m^2 \alpha_{n-1}$;
    •     $d_m = y_m + (\vartheta_{m-1}^2 \eta_{m-1}/\alpha_{n-1}) d_{m-1}$;
    •     $x_m = x_{m-1} + \eta_m d_m$;
    • If $x_m$ has converged: stop;
(c) Set $\rho_n = \tilde{r}_0^H w_{2n+1}$, $\beta_n = \rho_n/\rho_{n-1}$;
    $y_{2n+1} = w_{2n+1} + \beta_n y_{2n}$;
    $v_n = A y_{2n+1} + \beta_n (A y_{2n} + \beta_n v_{n-1})$.
```

The convergence check in step (2b) is usually based on the norm $\|r_m\|$ of the residual vector r_m corresponding to x_m. These vectors are not generated explicitly in Algorithm 5.1. However, we have the following upper bound available at no extra cost:

$$\|r_m\| \le \left\|W_{m+1}\Omega_{m+1}^{-1}\right\| \cdot \|f_{m+1} - H_m^{(e)} z_m\| \le \sqrt{m+1}\,\tau_m. \tag{5.9}$$

The inequalities in (5.9) follow from (3.11) (with $z = z_m$), (3.14), and

$$\left\|W_{m+1}\Omega_{m+1}^{-1}\right\| \le \sqrt{m+1}.$$

The latter estimate holds since, by (3.15), the columns of the $N \times (m+1)$ matrix $W_{m+1}\Omega_{m+1}$ all have unit length. Therefore, the convergence criterion in Algorithm 5.1 can be checked for the upper bound on the right-hand side of (5.9), and the actual norm $\|r_m\|$ only needs to be computed in the final stages of the iteration process.

Notice that the vectors w_{2n} in Algorithm 5.1 are not used directly; only their norms $\|w_{2n}\|$ are needed. We can eliminate the w_{2n}'s, and thus save one inner product per iteration, by approximating the weights ω_{2n} in (3.15) as follows. First, recall that by (3.3) and (2.3),

$$\|w_{2n+1}\| = \|r_n^{\mathrm{CGS}}\| = \left\|(\varphi_n(A))^2 r_0\right\|. \tag{5.10}$$

Since by (3.2), $w_{2n} = \varphi_n(A)\varphi_{n-1}(A) r_0$, the relation (5.10) suggests the approximation

$$\|w_{2n}\| \approx \sqrt{\|r_{n-1}^{\mathrm{CGS}}\| \cdot \|r_n^{\mathrm{CGS}}\|}.$$

This leads to the weighting strategy

$$\omega_m = \begin{cases} \|r_n^{\mathrm{CGS}}\| & \text{if } m = 2n+1 \text{ is odd,} \\ \sqrt{\|r_{n-1}^{\mathrm{CGS}}\| \cdot \|r_n^{\mathrm{CGS}}\|} & \text{if } m = 2n \text{ is even.} \end{cases} \tag{5.11}$$

In the case of (5.11) and, more generally, for any choice of weights $\omega_k > 0$ in (3.10) that does not require $\|w_{2n}\|$, we can eliminate w_{2n} in Algorithm 5.1. The resulting procedure is, except for the update of the different iterates in step (2b), identical with the CGS Algorithm 2.1, and it can be stated as follows.

Algorithm 5.2 (TFQMR Algorithm with General Weights $\omega_k > 0$).

```
(1) Start:
    (a) Choose $x_0 \in \mathbb{C}^N$;
    (b) Set $p_0 = u_0 = r_0^{\mathrm{CGS}} = r_0 = b - Ax_0$, $v_0 = Ap_0$, $d_0 = 0$;
            $\tau_0 = \omega_1$, $\vartheta_0 = 0$, $\eta_0 = 0$;
    (c) Choose $\tilde{r}_0$ such that $\rho_0 = \tilde{r}_0^H r_0 \neq 0$.
(2) For $n = 1, 2, \ldots,$ do:
    (a) Set $\sigma_{n-1} = \tilde{r}_0^H v_{n-1}$, $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$;
            $q_n = u_{n-1} - \alpha_{n-1} v_{n-1}$;
            $r_n^{\mathrm{CGS}} = r_{n-1}^{\mathrm{CGS}} - \alpha_{n-1} A(u_{n-1} + q_n)$;
    (b) For $m = 2n-1, 2n$ do:
        • Set $\vartheta_m = \omega_{m+1}/\tau_{m-1}$, $c_m = 1/\sqrt{1+\vartheta_m^2}$;
        •     $\tau_m = \tau_{m-1}\vartheta_m c_m$, $\eta_m = c_m^2 \alpha_{n-1}$;
        •     $d_m = y_m + (\vartheta_{m-1}^2 \eta_{m-1}/\alpha_{n-1}) d_{m-1}$,
        •         where $y_m = \begin{cases} u_{n-1} & \text{if } m = 2n-1, \\ q_n & \text{if } m = 2n; \end{cases}$
        •     $x_m = x_{m-1} + \eta_m d_m$;
        • If $x_m$ has converged: stop;
    (c) Set $\rho_n = \tilde{r}_0^H r_n^{\mathrm{CGS}}$, $\beta_n = \rho_n/\rho_{n-1}$;
            $u_n = r_n^{\mathrm{CGS}} + \beta_n q_n$;
            $p_n = u_n + \beta_n(q_n + \beta_n p_{n-1})$;
            $v_n = Ap_n$.
```

Finally, we note that x_n^{CGS} can be generated easily during the course of the TFQMR Algorithms 5.1 or 5.2, whenever a CGS iterate is desired. Indeed, in view of (3.19), (5.1), and (5.3), the CGS and TFQMR iterates are connected by

$$x_n^{\mathrm{CGS}} = x_{2n-1} + \alpha_{n-1} d_{2n}.$$

6. Numerical examples. We have performed numerical experiments with the CGS method, the Bi-CGSTAB algorithm, and the TFQMR algorithm with both weighting strategies (3.15) and (5.11). In this section, we present the results for two linear systems that are quite difficult to solve by iterative methods.

In both examples we used $x_0 = 0$ as starting vector, and $\tilde{r}_0$ was chosen as a vector with random entries from a normal distribution with mean 0.0 and variance 1.0. As stopping criterion, we used

$$\frac{\|r_n\|}{\|r_0\|} \leq 10^{-6} \tag{6.1}$$

for CGS and Bi-CGSTAB, and

$$\frac{\|r_m\|}{\|r_0\|} \leq 10^{-6} \tag{6.2}$$

for TFQMR. Note that CGS and Bi-CGSTAB both produce only one iterate x_n per iteration, while QMR generates two iterates x_m, with indices $m = 2n-1$ and $m = 2n$, in the nth step of the iteration. However, work and storage per iteration are roughly the same for all three methods. For both examples, we show the relative residual norm (6.1), respectively (6.2), plotted versus the iteration number n. The solid line is the convergence curve for the TFQMR Algorithm 5.1 (with weighting strategy (3.15)), the dotted line shows the behavior of CGS, and the dashed line is the Bi-CGSTAB convergence curve.

Example 6.1. We consider the partial differential equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + \gamma\left(x\frac{\partial u}{\partial x} + y\frac{\partial u}{\partial y}\right) + \beta u = f \quad \text{on} \quad (0,1)\times(0,1), \tag{6.3}$$

with Dirichlet boundary conditions. We discretize (6.3) using centered differences on a uniform 63×63 grid with mesh size $h = 1/64$. The resulting linear system has a sparse coefficient matrix A of order $N = 3969$. The parameters in (6.3) were chosen to be $\beta = -200$ and $\gamma = 100$. Note that this choice guarantees that the cell Reynolds number is smaller than one, and hence centered differences yield a stable discretization of (6.3). The right-hand side was chosen such that the vector of all ones is the exact solution of the linear system. The convergence behavior is shown in Fig. 6.1. For this example, we have also plotted (dashed-dotted line) the convergence curve for the TFQMR Algorithm 5.2 with the slightly less expensive weighting strategy (5.11). Note that the TFQMR curves for both strategies (3.15) and (5.11) are very close. This behavior is typical, and it was also observed in other numerical tests.

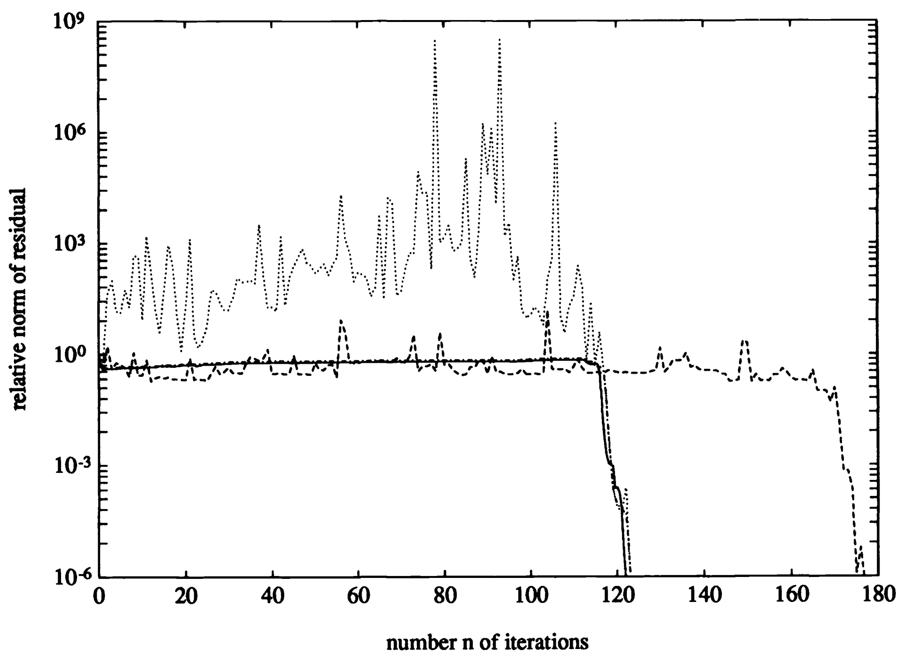


FIG. 6.1.

Example 6.2. This example was taken from the Harwell–Boeing set of sparse test matrices [2]. It is the fifth matrix from the SHERMAN collection, called SHERMAN5. It comes from a fully implicit black oil simulator on a $16 \times 23 \times 3$ grid, with three unknowns per grid point. The order of the matrix is 3312, and it has 20793 nonzero elements. The right-hand side b was also chosen as a vector with random entries. The convergence curves are shown in Fig. 6.2.

Both examples clearly demonstrate that the TFQMR method is certainly a remedy for the erratic convergence behavior of CGS. They also show that, in general, the residual norms for Bi-CGSTAB can still oscillate considerably, and that for difficult problems, convergence can be significantly slower than for CGS and TFQMR.

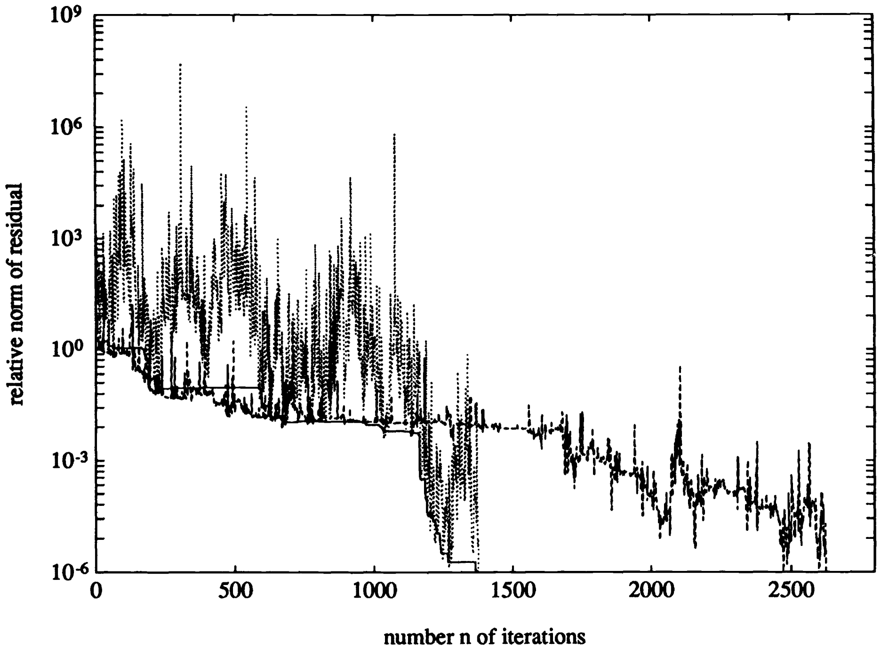


FIG. 6.2.

7. Conclusion. In this note, we proposed the TFQMR algorithm for solving general nonsingular non-Hermitian linear systems. The TFQMR method is closely related to the CGS algorithm, and it can be implemented by changing only a few lines in standard CGS. However, unlike CGS, the iterates of the TFQMR algorithm are characterized by a quasi minimization of the residual norm. This leads to smooth convergence curves for the TFQMR method, while CGS typically exhibits a rather irregular convergence behavior with wild oscillations in the residual norm.

Like standard CGS, the TFQMR algorithm described here can break down prematurely. Although these breakdowns are very rare in practice, for a robust implementation of the method that can be used as a black-box solver, it is crucial to incorporate look-ahead steps that allow to leap over those iterations in which exact breakdowns or near-breakdowns would occur in the standard process. The details of such a TFQMR algorithm with look-ahead will be presented elsewhere.

Finally, we remark that, based on the quasi-minimal residual property, one can derive upper bounds for the residual norms of the TFQMR iterates. Such a convergence result for the TFQMR method is given in [5, Thm. 6].

REFERENCES

[1] C. BREZINSKI AND H. SADOK, *Avoiding breakdown in the* CGS *algorithm*, Numer. Algorithms, 1 (1991), pp. 199–206.

[2] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[3] R. FLETCHER, *Conjugate gradient methods for indefinite systems*, in Proc. Dundee Conference on Numerical Analysis, 1975, Lecture Notes in Mathematics 506, G. A. Watson, ed., Springer-Verlag, Berlin, 1976, pp. 73–89.

[4] R. W. FREUND, *Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 425–448.

[5] ———, *Quasi-kernel polynomials and convergence results for quasi-minimal residual iterations*, in Numerical Methods of Approximation Theory, D. Braess and L. L. Schumaker, eds., Birkhäuser, Basel, 1992, to appear.

[6] R. W. FREUND, M. H. GUTKNECHT, AND N. M. NACHTIGAL, *An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices*, SIAM J. Sci. Statist. Comput., 14 (1993), pp. 137–158.

[7] R. W. FREUND AND N. M. NACHTIGAL, QMR: *a quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math., 60 (1991), pp. 315–339.

[8] R. W. FREUND AND H. ZHA, *Simplifications of the nonsymmetric Lanczos process and a new algorithm for solving symmetric indefinite linear systems*, Numerical Analysis Manuscript, AT&T Bell Laboratories, Murray Hill, NJ, 1992, in preparation.

[9] C. W. GEAR AND Y. SAAD, *Iterative solution of linear equations in* ODE *codes*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 583–601.

[10] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[11] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[12] 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.

[13] ———, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[14] 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.

[15] P. SONNEVELD, CGS, *a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[16] 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.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 483–505, March 1993

PRESERVING SYMMETRIES IN THE PROPER ORTHOGONAL DECOMPOSITION*

NADINE AUBRY[†], WEN-YU LIAN[†], AND EDRISS S. TITI[‡]

Abstract. The proper orthogonal decomposition (POD) (also called Karhunen–Loève expansion) has been recently used in turbulence to derive optimally fast converging bases of spatial functions, leading to efficient finite truncations. Whether a finite number of these modes can be used in numerical simulations to derive an "accurate" finite set of ordinary differential equations, over a certain range of bifurcation parameter values, still remains an open question. It is shown here that a necessary condition for achieving this goal is that the truncated system inherit the symmetry properties of the original infinite-dimensional system. In most cases, this leads to a systematic involvement of the symmetry group in deriving a new expansion basis called the symmetric POD basis. The Kuramoto–Sivashinsky equation with periodic boundary conditions is used as a paradigm to illustrate this point of view. However, the conclusion is general and can be applied to other equations, such as the Navier–Stokes equations, the complex Ginzburg–Landau equation, and others.

Key words. proper orthogonal decomposition, Karhunen–Loève expansion, empirical eigenfunctions, Kuramoto–Sivashinsky equation

AMS(MOS) subject classifications. 34C40, 35A40, 35K60, 58F32, 65M60

0. Introduction.

All computations involve a process of discretization in which infinite-dimensional (continuum) models, taking the form of partial differential equations (PDEs), are projected onto discrete or finite-dimensional forms. The way this finite projection is carried out is crucial for the accuracy and efficiency of the numerical simulation. Even if the process that is modeled is of low dimension, a finite projection may not be successful if it does not retain the essential dynamical features of the PDE such as the symplectic structure for Hamiltonian systems (see [18], [14], and [68]) or the preservation of dissipation in dissipative systems [29], [21], [38], [39]. In this paper, we are interested in reproducing the bifurcation diagram of dissipative equations based on the truncated proper orthogonal decomposition (POD) as defined below. In particular, we are concerned with deriving truncated dynamical systems that obey the symmetries of the original infinite-dimensional equation.

It is known that the long time behavior of certain PDEs such as the dissipative ones, is finite dimensional and sometimes relatively low dimensional. Moreover, it is sometimes possible to reduce the dynamics to a finite-dimensional system of ordinary differential equations (ODEs) in a rigorous manner. In the latter case, the reduced ODE is called an inertial form and the reduction takes place on a smooth finite-dimensional invariant manifold, which is called inertial manifold. More precisely, an inertial manifold is an invariant manifold that attracts all the trajectories exponentially fast and therefore it contains the global attractor [25]–[27]. The advantage is that stability and bifurcation calculations of the PDE can be performed for the inertial form [20], [38], [39]. In practice, an inertial manifold can be thought of as an

* Received by the editors July 1, 1991; accepted for publication (in revised form) June 22, 1992. This work was supported by Air Force Office of Scientific Research grant MSS89-57462, National Science Foundation grant DMS-8915672, and the United States Army Research Office through the Mathematical Sciences Institute, Cornell University. Part of this work was done while the third author was visiting the Mathematical Sciences Research Institute (MSRI).

[†] The Benjamin Levich Institute and the Mechanical Engineering Department, The City College of the City University of New York, New York, New York 10031.

[‡] Department of Mathematics, University of California, Irvine, California 92717, and Center for Applied Mathematics, Cornell University, Ithaca, New York 14853.

interaction law relating small and large scales, resembling an eddy viscosity model in turbulence [24]. The existence of an inertial manifold has been established for certain dissipative evolution PDEs such as the complex Ginzburg–Landau (CGL) equation, the Kuramoto–Sivashinsky equation (KSE), certain reaction-diffusion equations, the nonlocal Burgers equation, the Cahn–Hilliard equation, and others (see, e.g., [15] and references therein). Most recently, Kwak [44] has shown the existence of an inertial form for the two-dimensional Navier–Stokes equations with periodic boundary conditions. The difficulty, however, is that even when an inertial manifold is known to exist, it is often not unique and its exact form is unknown. Thus an approximation is necessary. Various approximation schemes have been developed to approximate inertial manifolds, or to directly approximate the global attractor (see, for example [20], [23], [27], [38], [39], [49], [66], and [67], and references therein).

Another approach to the issue of the finite projection of PDEs is the selection of "proper" modes, that is, modes that are relevant to the dynamics. These may be provided by the POD, also called Karhunen–Loève expansion, or empirical eigenfunctions method [45]–[48], [58], and [6]. The proper orthogonal decomposition of probability theory [45] has been developed, discussed, and extensively used to represent a random process. It has been proposed by Lumley [46]–[48] for identification of coherent structures in turbulence. It consists of decomposing a square integrable stochastic signal $u_\omega(x)$, where x belongs to the physical space X and ω is a random variable on the probability space Ω. For simplicity, we restrict ourselves to scalar functions, although the extension to vector-valued functions is straightforward. The proper orthogonal decomposition theorem [45] allows a decomposition of the random signal $u_\omega(x)$ into orthogonal deterministic functions $\varphi^n(x)$ and random coefficients a^n_ω :

$$u_\omega(x) = \sum_{n=1}^{\infty} a^n_\omega \varphi^n(x), \tag{1}$$

where the φ^n's are eigenfunctions, called in the literature POD modes or empirical eigenfunctions, of the following (Fredholm) integral equation:

$$\int_X R(x,x')\varphi(x')dx' = \lambda\varphi(x), \tag{2}$$

in which $R(x,x')$ denotes the two-(space)-point correlation function

$$R(x,x') = \langle u_\omega(x)u_\omega(x')\rangle_\omega, \tag{3}$$

that is, the ensemble average (over the random variable ω) of the product $u_\omega(x)u_\omega(x')$. It is well known (Hilbert–Schmidt and Mercer theorems; see, e.g., [56]) that if the correlation function is square integrable, then the set of eigenfunctions of the kernel $R(x,x')$ forms a complete set of $L^2(X)$. An important property of the POD is its optimally fast convergence in the sense that it maximizes the quantity $\langle (u_\omega,\varphi)^2\rangle_\omega$, where (.,.) denotes the canonical $L^2(X)$ scalar product, and $\|\varphi\|_{L^2} = 1$. On the other hand, the "energy" of the stochastic signal (defined as $\langle (u_\omega,u_\omega)\rangle_\omega$) is given by the sum of the eigenvalues λ (defined in (2)) counted with their multiplicities. Each eigenvalue, taken individually, represents the "energy" contribution of the corresponding eigenfunction. The method has been applied to various flows (see, for example, [9], [55], [50], [32], [33], [51], [57]–[61], [34], [16], [17], and references therein).

In the fluid mechanics literature, statistical stationarity and ergodicity are assumed (see, e.g., [6] and [58]). Indeed, under the assumption of the ergodicity of the flow on the global attractor, we can substitute the ensemble average by the time average, namely,

$$R(x,x') = \lim_{T\to\infty} \frac{1}{T} \int_0^T u(x,t)u(x',t)dt. \tag{4}$$

We stress the fact that, regardless of the ergodicity of the flow, we can still find a two-(space)-point correlation function using time averages, and find an orthonormal basis of $L^2(\mathrm{X})$ consisting of eigenfunctions of the correlation operator, defined in (4), provided the latter converges and satisfies the appropriate conditions (i.e., the conditions of Hilbert–Schmidt and Mercer theorems; see [56]). However, by implementing the long time average, as in (4), the "energy" contribution of the short-term events, such as the intermittent bursts in the presence of homoclinic orbits, might be very small. Therefore, we would not expect to be able to reproduce these events by taking only the largest "energy" POD modes in the Galerkin procedure. On the other hand, it is well known that, in certain cases, such short time events are responsible for an essential part of the chaotic dynamics. As a result, we might be obliged to take a larger Galerkin approximation, which will involve POD modes with almost zero "energy," in order to be able to reproduce the right dynamics. Indeed, a similar phenomenon can be observed in our computations in §4.2. The Galerkin approximation based on six POD modes, which contain 99.9995 percent of the "energy," could not reproduce the right dynamics. Only after taking seven modes are we able to get the right dynamics. This time-average procedure has been used by Aubry, Guyonnet, and Lima [4] to derive a space/time symmetric version of the expansion, the bi-orthogonal decomposition. This is a very useful tool to analyze space/time symmetries and bifurcations in complex spatio-temporal systems [2], [5], [7].

While the inertial manifold approach is global in the sense that such a manifold is invariant and contains the global attractor, the POD approach is local in phase space unless the global attractor is ergodic. Namely, the important POD modes, i.e., those with most of the "energy," span a linear manifold which closely approximates the ω-limit set of the particular trajectory that was used in the averaging process. The POD, indeed, leads to an optimally fast convergent expansion, in a certain sense, only for the particular signals $u(x,t)$ used in the determination of the two-(space)-point correlation function. Other signals, even those that are solutions of the same PDE for the same bifurcation parameter value, in particular those of other ergodic components of the global attractor, are likely to be poorly represented by the Galerkin approximation, based on a truncated set of these basis functions. In other words, spatial eigenfunctions derived from a particular solution necessarily provided a basis for $L^2(X)$, but a finite part of this basis may have absolutely no meaning for some components of the global attractor. Moreover, we emphasize that the characteristic spatial eigenfunctions, that is, those that have nonnegligible "energy," may completely change through bifurcations as the bifurcation parameter changes (see [2], [5], and [7]).

A finite-dimensional dynamical system has been derived by Aubry et al. [6] and Aubry and Sanghi [8], who projected the Navier–Stokes equation onto a finite number of basis spatial modes extracted from (2) and (4), where (4) had been previously experimentally determined. Low-dimensional dynamical systems thus derived could approximately reproduce the structure and dynamics of a turbulent flow in the vicinity of a wall. However, no parametric study with respect to Reynolds number was carried out and the global phase space study was not performed. One of the most important remaining issues is whether the POD method can be used to extract finite-dimensional approximations to a PDE in order to study global bifurcations. The aim is then to reconstruct the bifurcation diagram over a certain range of parameter values from a specific finite set of POD modes which would then be global on such a range. An attempt in this direction has been carried out by Sirovich and Rodriguez [57], [59] for the CGL equation. With three POD modes extracted from a chaotic solution, they could reproduce the period doubling route to chaos as first discovered by Keefe [40]. It is possible that a more careful bifurcation diagram analysis (including fixed points, precise bifurcation branch locations, etc.) may reveal problems of the type that will be pointed out in §4. (For a detailed global bifurcation analysis of the CGL equation, see, for instance, [12].) Along these lines, Deane et al. [16] have recently applied the POD method to the wake flow behind a cylinder at relatively small Reynolds number and observed similar difficulties. Finding a finite set of appropriate modes for the global bifurcation diagram is thus nontrivial. Although in this paper we do not answer this difficult question, we find a necessary condition that the finite set of global modes should satisfy. We are specifically interested in deriving Galerkin approximations that are inspired by the Karhunen–Loève procedure and that preserve the symmetric features of the problem and its global attractor (see Propositions 3 and 5 in §3).

The paper is organized as follows. In §1, we recall a few previous results concerning the KSE, which we need for our method implementation. In §2, we expand the solution of the equation by two complete sets of modes, the Fourier modes and the POD modes, and justify this operation in both cases. Then, in §3 we discuss the symmetries of the truncated finite sets of ODEs obtained by Galerkin projection of the PDE onto these modes. In the case of POD modes, we find a sufficient condition for the truncated system of ODEs which obeys the symmetries of the original PDE. This leads to a modification of the basic POD technique, by involving the average over the global symmetry group in the ensemble average. Finally, we show the necessity of this alternative procedure by studying the bifurcation diagram of stationary solutions of the KSE in §4. Moreover, in §4.2 we demonstrate that the modes with negligible "energy" are sometimes essential in reproducing the right dynamics, and hence must be included in the Galerkin projection. In conclusion, we raise the following questions: If it is not the "energy," then what should be the right criterion for determining the number of modes to be included in the Galerkin projection? What trajectories would be considered reliable to produce good PODs? And what should be the right ensemble average?

1. The Kuramoto–Sivashinsky equation.

As an example, we consider the KSE subject to periodic boundary conditions. This equation is a model for a variety of physical problems such as flame propagation and reaction-diffusion dynamics in combustion [62], [63]; thin film flows [10], [64], [35]; and, two-phase flows [30], [53]. It has been extensively studied analytically and numerically (see, for example, [38], [39], [31], [54], and references therein). Despite numerous bifurcations [41] and the complexity of some chaotic spatio-temporal solutions [36], [12], [13], [7], the KSE is known to have an inertial manifold [26], [27], [15] that is apparently low dimensional, as reflected by the numerical simulations. Foias et al. [20] and, Jolly, Kevrekidis, and Titi [38], [39] used the approximate inertial manifold technique to compute the bifurcation diagram in a relatively large range of the bifurcation parameter α (as defined below in (7)). Different approximate inertial manifolds and nonlinear Galerkin schemes were used and it was shown that, for a certain range of α, only three modes were enough to recover bifurcation branches. Along the same lines, Armbruster, Guckenheimer, and Holmes [1] used the center manifold technique to derive a four-dimensional ODE system to study local bifurcations around a critical parameter value.

We consider the normalized KSE

$$
\begin{aligned}
&\frac{\partial u}{\partial t}+\frac{\partial^4 u}{\partial x^4}+\frac{\partial^2 u}{\partial x^2}+u\frac{\partial u}{\partial x}=0, \qquad \text{where} \quad (x,t)\in R\times R^+,\\
&u(x,0)=u_0(x), \qquad \text{where} \quad x\in R,
\end{aligned} \tag{5}
$$

with periodic boundary conditions

$$
u(x,t)=u(x+L,t), \quad \text{with} \quad L>0, \tag{6}
$$

and

$$
\int_0^L u(x,t)dx=0.
$$

After rescaling time and space, (5) and (6) become

$$
\frac{\partial u}{\partial t}+4\frac{\partial^4 u}{\partial x^4}+\alpha\Big(\frac{\partial^2 u}{\partial x^2}+u\frac{\partial u}{\partial x}\Big)=0, \quad \text{where} \quad (x,t)\in R\times R^+, \tag{7}
$$

and where $\alpha = L^2/\pi^2$ is the bifurcation parameter. In this case, the boundary condition becomes

$$
u(x,t)=u(x+2\pi,t). \tag{8}
$$

In the following, we refer to (7) subject to the boundary condition (8) as the KSE. It is well known that the subspace of odd functions is an invariant subspace for this equation and the restricted system to this subspace is a dissipative system (see [52] and [65]). Recently, however, it was shown by Il'Yashenko [37] that the KSE is a dissipative system without the restriction to the odd case. Following the same proofs as in the odd case, and using the results of Il'Yashenko [37], we can show that the KSE has a compact global attractor and an inertial manifold. Moreover, since the space of solutions is invariant under translations $x \to x+\beta$ for every β (in the odd case β equals π only), namely, if $u(x,t)$ is a solution, then $u(x+\beta,t)$ is also a solution, the global attractor enjoys the same property. This translation symmetry is as crucial for the analysis performed in this paper as it was for the case in [1]. Generally, if the space of solutions of a dissipative system is invariant under a symmetry group G, then the global attractor will have the same property. Since certain dynamic behavior is structurally stable in the presence of symmetry, in general, we must ensure that the truncated system enjoys the symmetry properties of the nontruncated system.

2. Fourier modes and POD modes. In general, we can expand the solution of the KSE into a Fourier series

$$
u(x,t)=\sum_{j=1}^{\infty} a_j(t)\cos jx + b_j(t)\sin jx, \tag{9}
$$

where in the odd subspace, the a_j's are equal to zero. On the other hand, we can expand the solutions in terms of the eigenfunctions of the operator $\Re$ defined below, which will be referred to as POD modes. We denote by $\mathfrak{R} : L^2((0, 2\pi)) \to L^2((0, 2\pi))$ the operator

$$(\mathfrak{R}\varphi)(x) = \int_0^{2\pi} R(x, x')\varphi(x')dx',$$

where R is the two-(space)-point correlation function

$$R(x, x') = \lim_{T\to\infty} \frac{1}{T} \int_0^T u(x, t)u(x', t)dt, \tag{10}$$

and where $u(x, t)$ is a solution of the KSE. In general, there is no reason for the limit in (10) to converge. However, in real implementations we choose $u(x, t)$ for which the numerical limit in (10) exists. Hereafter, we will assume that the limit in (10) exists and that the correlation tensor is well defined.

PROPOSITION 1. *The eigenfunctions of* $\mathfrak{R}$, $\{\varphi_j\}$, *form a complete set of* $L^2((0, 2\pi))$.

Proof. It is clear that the operator $\mathfrak{R}$ is symmetric and nonnegative definite. Therefore, in order to apply the Hilbert–Schmidt and Mercer theorems (see, e.g., [56]), we must check that $R(x, x')$ is square integrable. The details of the proof are straightforward and based on both the regularity of the solutions of the KSE and the fact that it is a dissipative equation.

As a conclusion, every solution of the KSE can be expanded in terms of the φ_j's:

$$u(x, t) = \sum_{j=1}^{\infty} c_j(t)\varphi_j(x). \tag{11}$$

Remark 2. In view of the works of Il'Yashenko [37] and Foias and Témam [28], we can show that the solution of the KSE belongs to a Gevrey class of regularity. Therefore, we can implement the proof of Foias, Manley, and Sirovich [22] to show that the eigenvalues of the operator $\mathfrak{R}$ are bounded by an exponentially decaying function.

3. Symmetries.

It is expected that the Galerkin approximation based on the POD modes will converge optimally fast. However, the truncated system might not preserve the dynamical features of the original equations, such as dissipation (see [27], [21], [38], and [39]) or those originating from the symmetries. Since the space of solutions of the KSE is invariant under spatial translations, we are interested in deriving truncated dynamical systems that respect this property.

3.1. Symmetric POD modes.

We now seek conditions under which the POD eigenspaces enjoy the global symmetric properties of the solutions of the equation.

Suppose we have a PDE for which the space of solutions is invariant under a symmetry group G; namely, for every solution $u(x, t)$ of the PDE, $g \circ u(x, t)$ is also a solution (for every $g \in G$). The largest group with the above property will be called the global symmetry group. A symmetry group could be spatial, temporal, or spatio-temporal. In this paper, we will consider only spatial symmetry groups (for spatio-temporal symmetries, see, for instance, [5]). We call φ a symmetric POD mode if $g \circ \varphi$, for every $g \in G$, is an eigenfunction of the operator $\mathfrak{R}$ corresponding to the same eigenvalue. Each eigenspace of symmetric PODs is then invariant under the symmetry group G, and we call it a symmetric eigenspace. In Remark 4 (iv) we give some examples of particular solutions $u(x,t)$ for which the eigenspaces of the correlation functions, induced by $u(x,t)$, are symmetric eigenspaces. It is worth mentioning that if the flow is ergodic on the global attractor, then for almost every solution $u(x,t)$ in the global attractor, the POD eigenspaces of the correlation function, induced by $u(x,t)$, are symmetric eigenspaces (in this regard, see [11]). Since ergodicity is a rarity and is generally difficult to check, we propose a systematic method to produce symmetric POD modes by involving the average over the symmetry group in addition to the time average of the correlation function given by (10). In Proposition 3, we consider symmetry groups that are acting on the functions by changing variables on the physical space, such as reflections, translations, etc. (for spatio-temporal symmetries, see, for instance, [5]). We should mention that Sirovich [58] proposed using this idea to enlarge the data and generate more accurate POD modes, and has implemented it for this purpose in several applications (see, for example, [58], [61], and [17]).

However, our aim is fundamentally different. First, we mention that in many situations, the POD modes and the symmetric POD modes do not coincide. If we perform a data analysis only to carry out a local bifurcation study of the signal itself that provided the POD modes, the procedure of generating symmetric POD modes by averaging over the symmetry group is obviously not adequate because the POD modes prevent us from studying symmetry-related bifurcations. (See [5] for a full discussion and examples, and [42] and [43] for another approach using the PODs to study local bifurcation.) However, we emphasize that symmetric POD modes should necessarily be considered for the derivation of finite-dimensional dynamical systems that approximate the global dynamics of the equation (see Proposition 5). In general, this is an essential step, because, from the dynamical systems point of view, it is well known that certain dynamical features are structurally stable only in the presence of particular symmetries. Also, our computations in §4 demonstrate the importance of this step.

In this paper we consider only finite spatial symmetry groups. To be more specific, suppose that there is a symmetry group G consisting of N elements, such that for every $g \in G$, there is a one-to-one, invertible, and smooth function $\psi_g : D \to D$, where D is the domain in the physical space where the equation is defined, satisfying

$$\psi_{g\circ k}(x) = \psi_g(\psi_k(x)), \quad \psi_{g^{-1}}(x) = \psi_g^{-1}(x) \quad \text{and} \quad |\det(D\psi_g)| = 1. \tag{12}$$

Moreover, assume that if $u(x,t)$ is a solution to the PDE under consideration, then $g \circ u(x,t) = \alpha_g u(\psi_g(x),t)$, where α_g is a real number, is also a solution (i.e., the space of solutions is invariant under this kind of symmetry). Since G is a finite group we can easily show that α_g must be equal to either $+1$ or -1. We define the symmetric two-point correlation matrix as

$$R_s(x,x') = \lim_{T\to\infty} \frac{1}{NT} \int_0^T \sum_{g\in G} u(\psi_g(x),t)u(\psi_g(x'),t)dt, \tag{13}$$

and denote by $\mathfrak{R}_s$ the corresponding operator. Here again we assume that the limit in (13) exists.

PROPOSITION 3. *Under the above assumptions, if φ is an eigenfunction of $\mathfrak{R}_s$ with eigenvalues λ, then $\varphi(\psi_g(x))$ is also an eigenfunction corresponding to the same λ for every $g \in G$.*

Proof. Let φ be an eigenfunction of the operator $\mathfrak{R}_s$ so that it satisfies

$$\int_D R_s(x,x')\varphi(x')dx' = \lambda\varphi(x).$$

Using the definition of R_s in (13), and letting h be an arbitrary element of G, we can write

$$\begin{aligned}\int_D R_s(x,x')\varphi(\psi_h(x'))dx' &= \lim_{T\to\infty}\frac{1}{NT}\int_0^T\int_D\sum_{g\in G}u(\psi_g(x),t)u(\psi_g(x'),t)\varphi(\psi_h(x'))dx' \\ &= \lim_{T\to\infty}\frac{1}{NT}\int_0^T\int_D\sum_{g\in G}u(\psi_g(x),t)u(\psi_{g\circ h^{-1}}(y),t)\varphi(y)dy,\end{aligned}$$

where $y=\psi_h(x')$. Denoting $k=g\circ h^{-1}$, we obtain

$$\begin{aligned}\int_D R_s(x,x')\varphi(\psi_h(x'))dx' &= \lim_{T\to\infty}\frac{1}{NT}\int_0^T\int_D\sum_{k\in G}u(\psi_{k\circ h}(x),t)u(\psi_k(y),t)\varphi(y) \\ &= \lim_{T\to\infty}\frac{1}{NT}\int_0^T\int_D\sum_{k\in G}u(\psi_k(\psi_h(x)),t)u(\psi_k(y),t)\varphi(y)dy \\ &= \lambda\varphi(\psi_h(x)).\end{aligned}$$

Remark 4. (i) We remark that all the formal steps in the previous proof can be rigorously justified by using the regularity properties of the solution $u(x,t)$. Also note that for the same reason, $R(x,x')$ (and therefore $R_s(x,x')$) satisfies the conditions of Hilbert–Schmidt and Mercer theorems. Therefore, the number of linearly independent eigenfunctions in each eigenspace is finite. As a result, it is possible to choose an orthonormal basis $\{\varphi_j^s(x)\}_{j=1}^{\infty}$ of eigenfunctions of the operator $\mathfrak{R}_s$ such that $\mathfrak{R}_s\varphi_j^s = \lambda_j\varphi_j^s$, where $\lambda_1\geq\lambda_2\geq\cdots$, and any solution of the PDE can then be expanded as a series

$$u(x,t)=\sum_{j=1}^{\infty}c_j(t)\varphi_j^s(x).$$

(ii) In the case of the KSE with general periodic boundary conditions, the space of solutions is invariant under translations $T_\beta : x\to x+\beta$ for every β in $[0,2\pi]$. Therefore, it suffices to replace in Proposition 3 the summation $\sum_{g\in G}$ by the integral $\int_{[0,2\pi]}d\beta$ and to define $\psi_g(x)=\psi_\beta(x)=x+\beta$, where $\beta\in[0,2\pi]$. In this case, if $\varphi(x)$ is an eigenfunction of $\mathfrak{R}_s$ with eigenvalue λ, then $\varphi(x+\beta)$, for every $\beta\in[0,2\pi]$, is also an eigenfunction corresponding to the same λ. Moreover, here it can be easily shown that $R_s(x,x')$ is only a function of $x-x'$ and therefore the Fourier modes are eigenfunctions of $\mathfrak{R}_s$. However, they might not be ordered according to their wave numbers, because the mode with wave number 1, for instance, may not necessarily be the POD with maximum energy. As a result, the set of symmetric PODs in this case must coincide with the set of Fourier modes.

(iii) We observe that the subspace of odd solutions of the KSE is invariant under the Z_2 symmetry group generated by the translation $T_\pi : x\to x+\pi$. Note, also, that particular solutions may have additional symmetries, as we will see in §5.

(iv) If $R(x,x') = R_s(x,x')$, then the symmetric POD modes coincide with the regular ones, and in this case Proposition 3 is valid for the regular POD modes. This occurs, for instance, when the solution $u(x,t)$ used in the correlation function is itself invariant under the symmetry group (see [11]). Also, in the case of the translation symmetry group, the symmetric POD modes coincide with the regular ones in the cases of statistical homogeneity, or when the solution used in the correlation function is either a fixed point $u(x,t) = c$ or a traveling wave (in all three cases, R is only a function of $x - x'$). The case of traveling waves has been treated in detail in [5].

Next we show that the Galerkin projection of the KSE onto any finite number of symmetric eigenspaces gives an ODE system whose space of solutions is invariant under the translation symmetry group of the KSE. To illustrate our idea we consider the KSE restricted to the space of odd functions. However, we will discuss the general case as well as other equations in Remark 6.

Let $\{\varphi_j^s(x)\}_{j=1}^{\infty}$ be an orthonormal basis of symmetric eigenfunctions of $\mathfrak{R}_s$ for the KSE restricted to the space of odd functions. Let N satisfy $\lambda_{N+1} < \lambda_N$, where $\lambda_1 \geq \lambda_2 \geq \cdots$ are the eigenvalues of the operator $\mathfrak{R}_s$. Let

$$u_N(x,t) = \sum_{j=1}^{N} a_j(t)\varphi_j^s(x)$$

denote a solution to the Galerkin approximation of the KSE based on the φ_j^s's. It is clear from (7) that the $a_j(t)$'s satisfy the ODE system

$$\dot{a}_k + 4\sum_{j=1}^{N} A_{kj}a_j - \alpha\sum_{j=1}^{N} B_{kj}a_j + \alpha\sum_{j,m=1}^{N} C_{kjm}a_j a_m = 0 \quad \text{for } k = 1,2,\ldots,N, \tag{14}$$

where

$$\begin{aligned} A_{kj} &= \int_0^{2\pi} \varphi_j^{s''}(x)\varphi_k^{s''}(x)dx, \\ B_{kj} &= \int_0^{2\pi} \varphi_j^{s'}(x)\varphi_k^{s'}(x)dx, \\ C_{kjm} &= \int_0^{2\pi} \varphi_j^{s'}(x)\varphi_m^s(x)\varphi_k^s(x)dx, \end{aligned} \tag{15}$$

for $k,j,m = 1,2,\ldots,N$.

PROPOSITION 5. *Let* $\mathbf{a}(t) = (a_1(t),\ldots,a_N(t))^T$ *be a solution of* (14). *Denote by* $u_N(x,t) = \sum_{j=1}^N a_j(t)\varphi_j^s(x)$ *the corresponding solution to the Galerkin projection system. Then* $u_N(x+\pi,t) = \sum_{j=1}^N b_j(t)\varphi_j^s(x)$, *and* $\mathbf{b}(t) = (b_1(t),\ldots,b_N(t))^T$ *is a solution of* (14) (*i.e.,* $u_N(x+\pi,t)$ *is also a solution to the Galerkin projection*).

Proof. Because of Proposition 3 and Remark 4(i) there exists a matrix $\mathbf{\Gamma}$ such that for every $j = 1,\ldots,N$, we have

$$\varphi_j^s(x+\pi) = \sum_{k=1}^{N} \Gamma_{jk}\varphi_k^s(x). \tag{16}$$

Therefore,

$$\Gamma_{jk} = \int_0^{2\pi} \varphi_j^s(x+\pi)\varphi_k^s(x)dx = \int_0^{2\pi} \varphi_j^s(x)\varphi_k^s(x+\pi)dx = \Gamma_{kj}.$$

Moreover, we can easily check that $\mathbf{\Gamma}^2 = \mathbf{I}$. Hence,

$$\mathbf{\Gamma} = \mathbf{\Gamma}^T = \mathbf{\Gamma}^{-1}. \tag{17}$$

From (16) we can easily see that

$$u_N(x+\pi, t) = \sum_{j,k=1}^{N} a_j(t)\Gamma_{jk}\varphi_k^s(x) = \sum_{k=1}^{N} b_k(t)\varphi_k^s(x),$$

where $\mathbf{b}(t) = \mathbf{\Gamma}^T\mathbf{a}(t)$.

It remains to show that $\mathbf{b}(t)$ solves (14). From (15) it is easy to observe that

$$\mathbf{A} = \mathbf{A}^T, \quad \mathbf{B} = \mathbf{B}^T, \quad \mathbf{A} = \mathbf{\Gamma}\mathbf{A}\mathbf{\Gamma}^T, \quad \text{and} \quad \mathbf{B} = \mathbf{\Gamma}\mathbf{B}\mathbf{\Gamma}^T. \tag{18}$$

Next we show that

$$\sum_{j,k,m=1}^{N} \Gamma_{kl}C_{kjm}a_j a_m = \sum_{p,q=1}^{N} C_{lqp}b_q b_p \tag{19}$$

(the proof of (18) is similar and will be omitted). From (15) and because the functions $\varphi_m^s(x)$ are periodic of period 2π, we have

$$C_{kjm} = \int_0^{2\pi} \varphi_j^{s\prime}(x+\pi)\varphi_m^s(x+\pi)\varphi_k^s(x+\pi)dx;$$

by (16),

$$\begin{aligned} C_{kjm} &= \int_0^{2\pi} \sum_{i,p,q=1}^{N} \Gamma_{jq}\varphi_q^{s\prime}(x)\Gamma_{mp}\varphi_p^s(x)\Gamma_{ki}\varphi_i^s(x)dx \\ &= \sum_{i,p,q=1}^{N} \Gamma_{jq}\Gamma_{mp}\Gamma_{ki}C_{iqp}. \end{aligned}$$

From (14) and the above, we obtain

$$\begin{aligned} \sum_{j,m=1}^{N} C_{kjm}a_j a_m &= \sum_{i,p,q,j,m=1}^{N} \Gamma_{jq}\Gamma_{mp}\Gamma_{ki}C_{iqp}a_j a_m \\ &= \sum_{i,p,q=1}^{N} \Gamma_{ki}C_{iqp}(\mathbf{\Gamma}^T\mathbf{a})_q(\mathbf{\Gamma}^T\mathbf{a})_p. \end{aligned}$$

Hence, because of (17) and the above, we have

$$\begin{aligned} \sum_{j,m,k=1}^{N} \Gamma_{kl}C_{kjm}a_j a_m &= \sum_{j,m,k=1}^{N} \Gamma_{lk}C_{kjm}a_j a_m \\ &= \sum_{p,q=1}^{N} C_{lqp}(\mathbf{\Gamma}^T\mathbf{a})_q(\mathbf{\Gamma}^T\mathbf{a})_p, \end{aligned}$$

which proves (19).

To conclude our proof, we apply $\mathbf{\Gamma}^T$ to (14) and use (17), (18), and (19).

Remark 6. (i) The above proof can be easily extended to other equations with polynomial-type nonlinearities, and other kinds of symmetry, such as reflection. To be more specific, we can prove results similar to those of Proposition 5 for the CGL equation with periodic boundary conditions, and to the results of problems considered in [58], [17], and [16].

(ii) It is clear from the above proof (in particular, see (16)) that if the KSE is projected onto the Fourier modes, the space of solutions of the truncated system in this case is invariant with respect to the translation symmetry.

3.2. Dissipation and POD modes. Showing the dissipation property for the KSE is a little tricky, unlike other dissipative equations, such as the Navier–Stokes equations or the CGL equation (see [52] and [37]). Without knowing the spatial structure of the first N POD modes it will not be easy to show that the truncated system (14) is dissipative. In fact, if N is not large enough, depending on α, the system (14) might not be dissipative at all. Moreover, in the latter case, some of the solutions might even blow up in finite time. This issue of dissipativity of the discretized KSE has been discussed in detail for various numerical schemes in [38], [39], [21], and [27]. However, let us remark that in the cases of the Navier–Stokes equations or the CGL equation, for instance, the dissipation property always holds independently of the orthonormal basis with respect to which we truncate.

4. Computations of the Kuramoto–Sivashinsky equation. In this section we derive sets of ODEs via a Galerkin projection of the KSE onto both Fourier and POD modes. In order to demonstrate our point of view, it will be more instructive if we restrict our computations to the invariant subspace of odd functions. Otherwise, in view of Remark 4(ii), the symmetric PODs for the KSE will coincide with the Fourier modes. It is worth mentioning that certain interesting solutions, such as traveling waves, will not be allowed in the odd case. Nevertheless, the dynamics of the restricted KSE is still rich, as indicated in the computations of Jolly, Kevrekidis, and Titi [38], [39]. In this work, we are specifically interested in the bifurcation diagram for parameter values $0 \leq \alpha \leq 40$. In the following, the bifurcation diagrams are obtained by using the bifurcation package AUTO [19].

4.1. Projection onto Fourier modes. We first integrate a set of 14 ODEs derived from Fourier modes, as the bifurcation diagram thus obtained is qualitatively correct in the parameter range $0 \leq \alpha \leq 40$ for both steady and unsteady solutions. Indeed, by increasing the number of Fourier modes, we could not notice any major difference. As first noted by Brown [12] and Jolly, Kevrekidis, and Titi [38], we could even decrease the number of Fourier modes to 6, but 14 were kept for the computations described below. The steady-state bifurcation diagram is plotted in Fig. 1 from a set of 6 ODEs based on Fourier modes, and it will be our reference. (This diagram is identical to that obtained by Jolly, Kevrekidis, and Titi [38].)

We now recall in detail the symmetries of various branches since, as explained earlier, they play an essential role for the implementation of the POD modes (see, e.g., [41], [38], and references therein). First, for small values of $\alpha, \alpha < 4$, the trivial solution is asymptotically stable. A stability analysis of this trivial fixed point shows that a pitchfork bifurcation occurs at $\alpha = 4$ and that subsequent bifurcations take place on the trivial branch, from which primary steady branches emerge. It is fundamental to note that since the linear part of the KSE is diagonal, linearized modal equations decouple and single modes bifurcate: $\sin x$ at $\alpha = 4$, $\sin 2x$ at $\alpha = 16$,

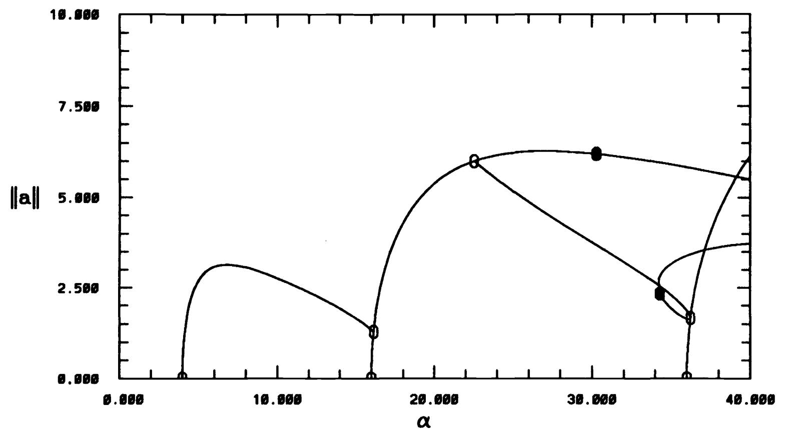


FIG. 1. *Steady-state bifurcation diagram of a dynamical system derived by projecting the* KSE *restricted to the odd subspace onto a set of six Fourier modes. The open circles represent steady-state bifurcation points, and the black circles represent Hopf bifurcations.*

$\sin 3x$ at $\alpha = 32$, $\sin 4x$ at $\alpha = 64$, etc. For this reason, branches thus born are called unimodal, bimodal, trimodal, or quadrimodal branches since young solutions (close to their birth) have one, two, three, or four humps, respectively. Then, by nonlinear quadratic interactions, multiple frequencies are activated so that solutions on an n-modal branch are linear combinations of $\sin jx$, $j = 1, 2, 3, \ldots$, and have a period of $2\pi/n$. Then secondary branches of various periods emerge from primary branches on which solutions may have various periods. In the parameter range $0 \le \alpha \le 40$, a secondary branch connects the bimodal and the trimodal branches before turning back to eventually go to the quadrimodal branch. The part connecting the bimodal and the trimodal branches is called the bitrimodal branch and has periodicity of 2π.

Now we remark that the global symmetry group of the KSE reduced to the odd subspace is the discrete Z_2 symmetry group, i.e., if $u(x,t)$ is a solution then $u(x+\pi,t)$ is also a solution. This implies that steady-state branches occur in pairs, both branches of the same pair being superposed in the bifurcation diagram. This is due to the fact that symmetric solutions are only distinct by a phase shift and therefore they have the same L^2 norm. Both symmetric odd-modal branches have the same stability, while both symmetric even-modal branches have different stabilities [41], [38].

4.2. Computation of POD modes. Let us recall that a Hopf bifurcation occurs on one of the bimodal branches at the parameter value $\alpha = 30.278$ (see [38]). The limit cycle thus born is stable and is invariant under the spatial and temporal symmetries $\widetilde{T} : t \to \tau/2$ and $T_\pi : x \to x + \pi$, so that $u(x,t) = u(x+\pi, t+\tau/2)$, where τ is the limit cycle (temporal) period. These symmetries have consequences on the POD modes and their coefficients that we do not exploit here [5], [7]. We remark that the symmetry of the bimodal branch $x \to x + \pi/2$ is lost, due to the fact that the Hopf bifurcation occurs on one of the two bimodal stationary branches only. The cycle has, however, the global symmetry of the KSE. At $\alpha = 32.85$, there is a symmetry-breaking bifurcation. The symmetric cycle becomes unstable while two stable asymmetric limit cycles appear in the phase space, symmetric one to the other through the symmetry $x \longrightarrow x + \pi$. Each asymmetric limit cycle then undergoes a period-doubling bifurcation sequence, leading to an apparently chaotic solution. (See [38] for a description of the whole bifurcation series from the bimodal branch. Also see [13] and [7] for a spatio-temporal analysis of stable solutions in this bifurcation series.) POD modes have been computed from such an apparently chaotic solution at the parameter value $\alpha = 33.005$. More precisely, Fourier coefficients of the POD modes are computed from the Fourier coefficients of the solution since we have

$$\sum_{m=1}^{N} R_{nm}\varphi_{mj} = \lambda_j \varphi_{nj}, \qquad n, j = 1, 2, \ldots, N,$$

where

$$\varphi_j(x) = \sum_{n=1}^{N} \varphi_{nj} \frac{\sin nx}{\sqrt{\pi}},$$

and

$$R_{ij} = \lim_{T \to \infty} \frac{1}{T} \int_0^T c_i(t) c_j(t) dt, \qquad 1 \le i \le N, 1 \le j \le N,$$

where c_k stands for the k Fourier coefficient of the solution $u_N(x,t)$, i.e., $u_N(x,t) = \sum_{n=1}^{N} c_n(\sin nx/\sqrt{\pi})$. Recall that in our computations $N = 14$.

In the computation of the apparently chaotic solution, the timestep used and time integration are 0.005 and 20, respectively (after 20 steps the time average was stabilizing), while the initial conditions were taken to be $c_1 = -0.9518, c_2 = 5.9054, c_3 = 5.9296, c_4 = -0.6256, c_5 = -0.9899, c_6 = -0.2159$, and $c_i = 0$ for all $i > 6$.

We denote by p_k the eigenvalue normalized by the energy

$$p_k = \frac{\lambda_k}{\sum_{i=1}^{N} \lambda_i},$$

and by S_k the partial energy

$$S_k = \sum_{i=l}^{k} \lambda_i.$$

The value of λ_k, p_k, and S_k are reported in Table 1. Fourier coefficients of POD modes are listed in Table 2. These tables show that the first two modes contain 96 percent of the energy and the first three modes, 99 percent of the energy. Although the first, second, and third POD modes get their main contribution from the second, third, and first Fourier modes, respectively, the other Fourier coefficients of the POD modes are not all zero: for example, φ_1 has nonnegligible first four Fourier coefficients.

First, we mention that the integration of the dynamical system retaining seven or more POD modes using the same initial conditions as those used in the dynamical system based on Fourier modes leads to a good reproduction of the apparently chaotic trajectory. However, we could not obtain a stable orbit of this type by reducing the number of POD modes to six or less: the trajectory was attracted by a stable fixed point. Although the quasi totality of the energy, namely 99.9995 percent, is included

TABLE 1
List of eigenvalues.

Index of POD mode k	Eigenvalue λ_k	Normalized eigenvalue p_k	Partial energy S_k
1	0.814592E+01	0.526481E+00	0.526481E+00
2	0.671891E+01	0.434251E+00	0.960732E+00
3	0.448952E+00	0.290163E−01	0.989748E+00
4	0.151746E+00	0.980755E−02	0.999556E+00
5	0.666166E−02	0.430551E−03	0.999986E+00
6	0.136265E−03	0.880696E−05	0.999995E+00
7	0.665096E−04	0.429860E−05	0.100000E+01
8	0.614423E−05	0.397109E−06	0.100000E+01
9	0.940076E−07	0.607583E−08	0.100000E+01
10	0.216713E−07	0.140065E−08	0.100000E+01
11	0.227465E−09	0.147014E−10	0.100000E+01
12	0.301249E−10	0.194701E−11	0.100000E+01
13	0.886460E−12	0.572930E−13	0.100000E+01
14	0.419195E−13	0.270931E−14	0.100000E+01

TABLE 2
List of the first 8 POD *modes computed from* 14 *Fourier modes. There is one* POD *mode per column. Its* 14 *Fourier coefficients are listed from top to bottom following the orders of the corresponding wave numbers.*

	1	2	3	4	5	6	7	8
1	0.0434	−0.3582	0.9293	0.0514	−0.0586	0.0041	−0.0104	0.0020
2	0.9881	0.0824	−0.0202	0.1164	0.0156	0.0511	−0.0046	0.0060
3	−0.0766	0.9190	0.3634	0.0437	0.1227	0.0079	0.0227	−0.0036
4	−0.1249	−0.0412	−0.0603	0.9472	0.0601	0.2759	−0.0254	0.0393
5	0.0041	−0.1364	0.00140	−0.0669	0.9617	0.0297	0.2223	−0.0315
6	−0.0164	0.0052	0.0115	−0.2816	−0.0207	0.9262	−0.0830	0.2346
7	0.0010	0.0060	−0.0024	0.0132	−0.2269	0.1033	0.9407	−0.0791
8	0.0028	0.0001	−0.0005	0.0284	0.0156	−0.2240	0.1060	0.9525
9	−0.0002	0.0007	0.0002	−0.0010	0.0259	−0.0203	−0.2133	0.0039
10	−0.0002	−0.0001	0.0000	−0.0010	−0.0019	0.0347	−0.0111	−0.1691
11	0.0000	−0.0001	0.0000	0.0000	−0.0013	0.0015	0.0244	−0.0005
12	0.0000	0.0000	0.0000	−0.0001	0.0001	−0.0032	0.0003	0.0154
13	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	−0.0013	0.0001
14	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	−0.0006

in the first six modes, these modes do not seem to be sufficient to reproduce the dynamical system. In this regard, it is worth mentioning that we have found that the stability of the cycles appearing through the period-doubling sequence, as well as that of the apparently chaotic regime it leads to, is very sensitive to the precision of the numerical scheme. Thus it is not surprising that it is sensitive to the presence of high modes. We now return to the reconstruction of the bifurcation diagram of stationary solutions. Obviously, if the number of modes retained in the Galerkin projection is equal to the number of Fourier modes, the bifurcation diagram is exactly the same as that found with Fourier modes. Discrepancy starts appearing as the number of POD modes is decreased. In a first step, we are interested in reconstructing the steady-state bifurcation diagram for parameter values below $\alpha = 33.005$, at which the apparently chaotic solution has been computed for POD mode extraction. Bifurcation diagrams obtained with four, five, six, and seven POD modes are presented in Figs. 2, 3, 4, and 5, respectively. As we compare these diagrams with our reference (Fig. 1), discrepancies can be immediately observed, such as the parameter values of the steady-state bifurcations from the trivial branch when only four POD modes are retained. In particular, this shows that although four POD modes account for 99.96 percent of the total energy of the apparently chaotic solution at $\alpha = 33.005$, they are not sufficient for a correct reproduction of the full dynamical system. More importantly, a striking feature of Figs. 2 and 3 is that all branches appear in pairs, but each pair is not superposed as it should be by symmetry considerations. This decoupling has secondary consequences: The steady-state bifurcation at the intersection between the unimodal and bimodal branches and that at the intersection between the bimodal and secondary branch have disappeared. Branch decoupling still exists, but is less visible in Figs. 4 and 5, while the inexistence of the steady-state bifurcation point connecting the unimodal and the bimodal branches persists.

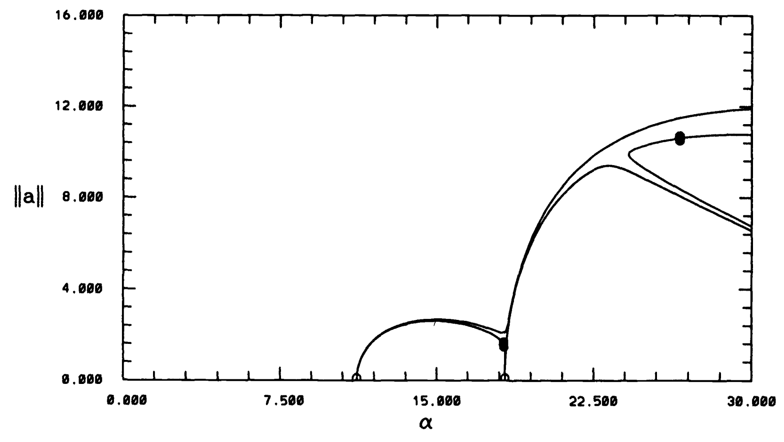


FIG. 2. *Steady-state bifurcation diagram of a dynamical system derived by projecting the* KSE *restricted to the odd invariant subspace onto the set of the first four* POD *modes extracted from an apparently chaotic solution at* $\alpha = 33.005$.

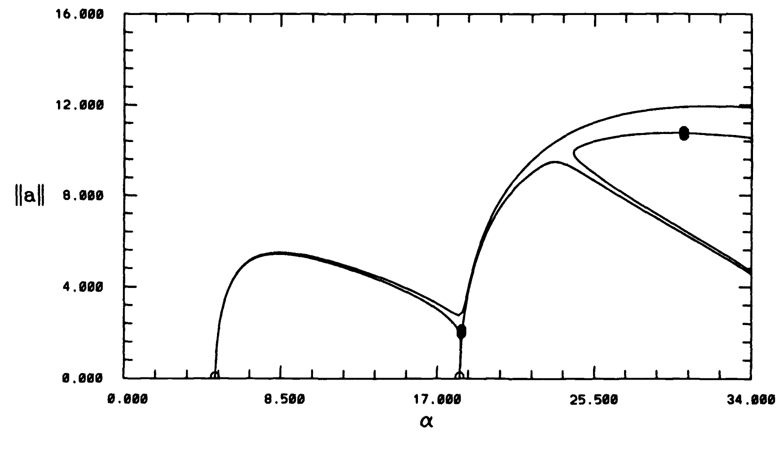


FIG. 3. *Same as Fig.* 2 *with the first five* POD *modes.*

The solutions of the truncated dynamical systems that we have derived are thus not invariant through the Z_2 symmetry $x \to x + \pi$ under which the space of solutions of the KSE is invariant. Therefore, in view of the work of Berkooz [11], this suggests that the apparently chaotic flow around $\alpha = 33.005$ is not ergodic. Hence, in order to impose this symmetry in the absence of ergodicity, we now compute the symmetric POD modes, as we previously proposed in §3. This is equivalent to taking into account the chaotic solution already considered and its symmetric companion. Symmetric POD modes and their respective eigenvalues are shown in Tables 3 and 4, respectively.

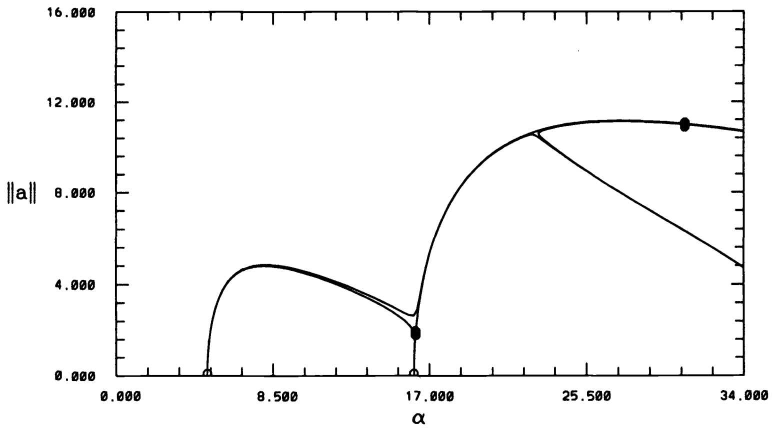


FIG. 4. *Same as Fig.* 2 *with the first six* POD *modes.*

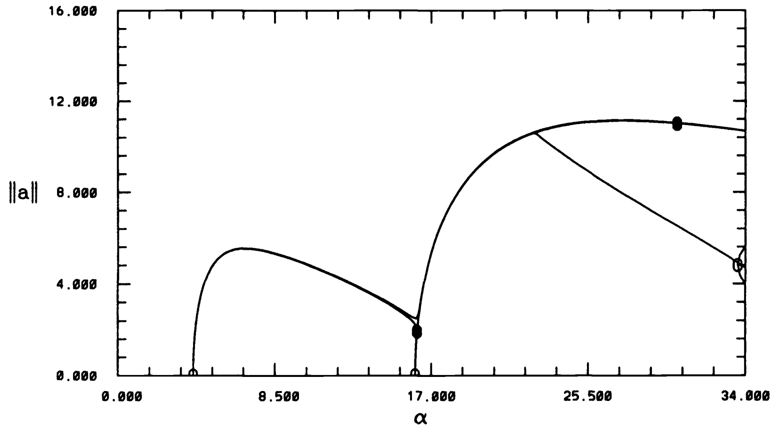


FIG. 5. *Same as Fig.* 2 *with the first seven* POD *modes.*

We note that the convergence given by the partial energies S_k is almost as fast as in the first case (Table 1) without symmetry involvement. An essential difference occurs among the eigenfunctions φ_k, which are now linear combinations of $\sin jx$,

TABLE 3
List of symmetric eigenvalues.

Index of POD mode k	Eigenvalue λ_k	Normalized eigenvalue p_k	Partial energy S_k
1	0.813340E+01	0.525672E+00	0.525672E+00
2	0.672331E+01	0.434536E+00	0.960207E+00
3	0.448948E+00	0.290160E−01	0.989223E+00
4	0.158653E+00	0.102539E−01	0.999477E+00
5	0.782172E−02	0.505527E−03	0.999983E+00
6	0.191528E−03	0.123787E−04	0.999995E+00
7	0.679747E−04	0.439329E−05	0.999999E+00
8	0.796359E−05	0.514697E−06	0.100000E+01
9	0.107020E−06	0.691684E−08	0.100000E+01
10	0.319819E−07	0.206703E−08	0.100000E+01
11	0.302034E−09	0.195208E−10	0.100000E+01
12	0.437756E−10	0.282927E−11	0.100000E+01
13	0.148950E−11	0.962684E−13	0.100000E+01
14	0.588551E−13	0.380387E−14	0.100000E+01

TABLE 4
List of the first 8 POD *modes computed from* 14 *Fourier modes. Each column corresponds to one* POD *mode. Their* 14 *Fourier coefficients are listed from top to bottom following the orders of the corresponding wave numbers.*

	1	2	3	4	5	6	7	8
1	0.0000	−0.3611	0.9306	0.0000	−0.0589	0.0000	−0.0097	0.0000
2	0.9917	0.0000	0.0000	0.1180	0.0000	0.0508	0.0000	0.0068
3	0.0000	0.9225	0.3659	0.0000	0.1208	0.0000	0.0230	0.0000
4	−0.1276	0.0000	0.0000	0.9526	0.0000	0.2726	0.0000	0.0433
5	0.0000	−0.1362	0.0106	0.0000	0.9652	0.0000	0.2214	0.0000
6	−0.0160	0.0000	0.0000	−0.2790	0.0000	0.9285	0.0000	0.2446
7	0.0000	0.0059	−0.0018	0.0000	−0.2228	0.0000	0.9503	0.0000
8	0.0028	0.0000	0.0000	0.0273	0.0000	−0.2447	0.0000	0.9557
9	0.0000	0.0007	0.0002	0.0000	0.0245	0.0000	−0.2161	0.0000
10	−0.0002	0.0000	0.0000	-0.0009	0.0000	0.0343	0.0000	−0.1571
11	0.0000	−0.0001	0.0000	0.0000	−0.0011	0.0000	0.0247	0.0000
12	0.0000	0.0000	0.0000	−0.0001	0.0000	−0.0027	0.0000	0.0127
13	0.0000	0.0000	0.0000	0.0000	−0.0001	0.0000	−0.0014	0.0000
14	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	−0.0003

where all the j's are either odd or even. Note that each one is invariant under the translation symmetry $x \to x+\pi$. Note that they were almost, but not quite, invariant in the previous case. (This small discrepancy was enough to make the POD modes nonsymmetric, which was reflected in the unimodal branch decoupling). Symmetric POD modes are therefore invariant under the global symmetry and, in this case, either $\varphi_k(x) = \varphi_k(\psi_g(x))$ or $-\varphi_k(x) = \varphi_k(\psi_g(x))$ (with $\psi_g(x) = x$ or $\psi_g(x) = x+\pi$) so that the eigenvalues are not degenerate. It immediately follows that any truncated system based on these modes is necessarily invariant under the symmetry group generated by T_π (see Proposition 5). This is immediately checked by reconstructing the steady-state bifurcation diagram: The pair of unimodal branches are now superposed as they should be and a steady-state bifurcation occurs when the pair meets with the bimodal branches. The same is true for the pair of bitrimodal branches that are now superposed and meet one bimodal branch through a steady-state bifurcation. This can be seen on bifurcation diagrams in Figs. 6, 7, 8, and 9, where four, five, six, and seven symmetric POD modes have been retained. However, the decoupling of the two bimodal branches still persists, due to the fact that the symmetry involved here is specific to the branch $(T_{\pi/2})$. Note that this symmetry cannot be imposed on physical grounds in our dynamical systems, as the global symmetry could. However, it tends to disappear as the dimension of the dynamical system increases; with six POD modes, it is no longer noticeable. The bifurcation diagram using six POD modes is qualitatively correct. However, we should recall that a truncated dynamical system based on six Fourier modes also gives good results. Therefore, we have not been able to decrease the number of necessary modes in the truncated dynamical system by taking POD modes, as we might have expected, not to mention that by using approximate inertial manifolds, three Fourier modes were enough to give the right qualitative diagram [20], [38], [39].

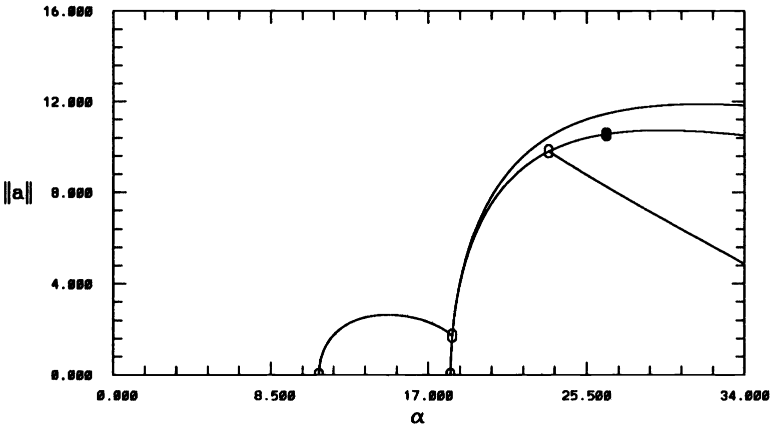


FIG. 6. *Same as Fig.* 2 *with the first four symmetric* POD *modes.*

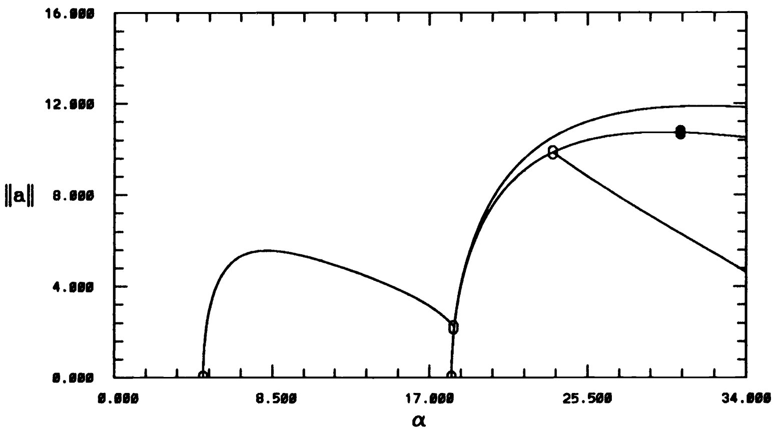


FIG. 7. *Same as Fig.* 2 *with the first five symmetric* POD *modes.*

5. Conclusion and discussion. We have proposed a technique to derive finite sets of ODEs based on POD modes that preserves the symmetry of the infinite-dimensional system. It consists of systematically involving the symmetry group in

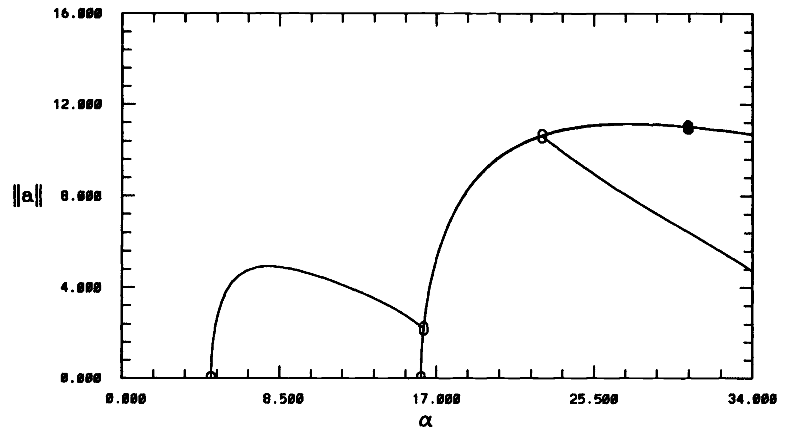


FIG. 8. *Same as Fig.* 2 *with the first six symmetric* POD *modes.*

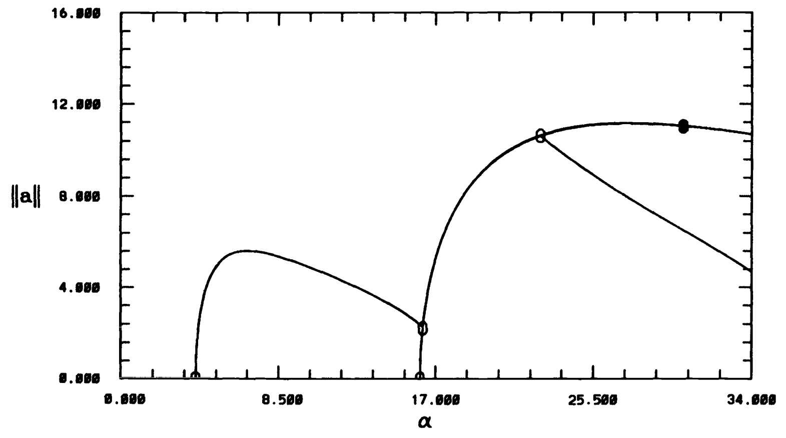


FIG. 9. *Same as Fig.* 2 *with the first seven symmetric* POD *modes.*

the derivation of basis modes. We have applied the technique to the KSE subject to periodic boundary conditions and restricted to the odd invariant subspace. However, we were not able to decrease the number of necessary POD modes below six, which is also the minimum number of Fourier modes necessary for the reconstruction of a qualitatively correct steady-state bifurcation diagram. POD modes that we have found numerically are not identical to Fourier modes. Even the first modes accounted for higher Fourier components, which are perhaps responsible for introducing extra dissipation in our dynamical system compared to a dynamical system based on Fourier modes. This could be the reason why the unbounded branches found by Jolly, Kevrekidis, and Titi [38], [39] do not occur as often in low-dimensional dynamical systems based on POD modes. The counterpart of this is perhaps a misrepresentation of low Fourier components in low-order POD modes. This did not have a major effect for values of parameters below 33, but it may have noticeable consequences on the trimodal branch on which the first steady-state bifurcation disappears when six (symmetric or nonsymmetric) POD modes are retained (see Fig. 10). Although our work focuses on the symmetry issue, we point out that this may not be the only problem in the procedure. Again, we recall that POD modes may completely change as the bifurcation parameter varies.

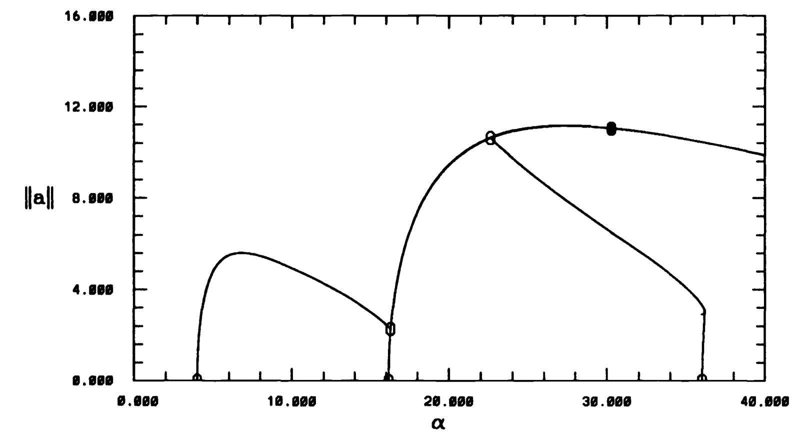


FIG. 10. *Same as Fig. 9 for a broader range of parameter values* $(0 < \alpha < 40)$.

To conclude, we point out that it is not trivial, at least for the PDE chosen as an example in this paper, to derive global modes that would be superior to Fourier modes and valid for a finite projection of the ODE over a relatively broad range of parameter values. This is not surprising since characteristic POD modes can dramatically change through bifurcations. From this point of view, local studies in phase space might be more promising [5], [7], [42], [43].

Acknowledgment. The authors would like to thank Professor I.G. Kevrekidis for the interesting discussions.

REFERENCES

[1] D. ARMBRUSTER, J. GUCKENHEIMER, AND P. HOLMES, *Kuramoto-Sivashinsky dynamics on the center-unstable manifold*, SIAM J. Appl. Math., 49 (1989), pp. 676–691.
[2] N. AUBRY, *On the hidden beauty of the proper orthogonal decomposition*, Theoret. Computational Fluid Dynamics, 2 (1991), pp. 339–352.
[3] N. AUBRY, M. P. CHAUVE, AND R. GUYONNET, *Transition to turbulence on a rotating flat disk*, Phys. Fluids A, submitted.
[4] N. AUBRY, R. GUYONNET, AND R. LIMA, *Spatio-temporal analysis of complex signals: Theory and applications*, J. Statist. Phys., 64 (1991), pp. 683–738.
[5] ———, *The bi-orthogonal decomposition and spatio-temporal bifurcations involving symmetries*, Nonlinear Sci., 2 (1992), pp. 183–215.
[6] N. AUBRY, P. HOLMES, J. L. LUMLEY, AND E. STONE, *The dynamics of coherent structures in the wall region of a turbulent boundary layer*, J. Fluid Mech., 192 (1988), pp. 115–173.

[7] N. Aubry and W. Y. Lian, *Analysis of spatio-temporal complexity in the Kuramoto–Sivashinsky equation*, Preprint # 9106019, Levich Institute, City College of the Univ. of New York, NY, 1991.

[8] N. Aubry and S. Sanghi, *Bifurcations and bursting of streaks in the wall layer*, in Proc. Grenoble (France) Conf. Organized Structures and Turbulence in Fluid Mechanics, Sept. 18–21, 1989, O. Metais and M. Lesieur, eds., Kluever Publishers, Dordrecht, the Netherlands, 1991, pp. 227–251.

[9] H. P. Bakewell and J. L. Lumley, *The viscous sublayer and adjacent wall region in turbulent pipe flows*, Phys. Fluids, 10 (1967), pp. 1880–1889.

[10] D. J. Benney, *Long wave in liquid films*, J. Math. Phys., 45 (1966), pp. 150–155.

[11] G. Berkooz, *Observations on the proper orthogonal decomposition*, in Studies in Turbulence, Proc. Lumley Symposium, S. T. B. Gatski, S. Sarkar, and C. Speziale, eds., Springer-Verlag, New York, 1991.

[12] H. S. Brown, *A computer assisted, nonlinear dynamic study of instabilities and pattern formation for interfacial waves*, Ph. D. thesis, Chemical Engineering Dept., Princeton Univ., Princeton, NJ, 1991.

[13] H. S. Brown, M. S. Jolly, and I. G. Kevrekidis, *A minimal model for spatio-temporal patterns in thin film flow*, Patterns and Dynamics in Reactive Media, IMA Volumes on Mathematics and its Applications, 37, H. Swinney, R. Aris, and D. G. Aronson, eds., Springer-Verlag, New York, 1991.

[14] P. J. Channel and C. Scovel, *Symplectic integration of Hamiltonian systems*, Nonlinearity, 3 (1990), pp. 231–259.

[15] P. Constantin, C. Foias, B. Nicolaenko, and R. Témam, *Integral Manifolds and Inertial Manifolds for Dissipative Partial Differential Equations*, Applied Mathematics Sciences, 70, Springer-Verlag, Berlin, 1988.

[16] A. E. Deane, I. G. Kevrekidis, G. E. Karniadakis, and S. A. Orszag, *Low dimensional models for complex geometry flows: Application to grooved channels and circular cylinders*, Phys. Fluids A, 3 (1991), pp. 2337–2354.

[17] A. E. Deane and L. Sirovich, *A computational study of the Rayleigh–Bénard convection. Parts* 1 *and* 2, J. Fluid Mech., 222 (1991), pp. 231–265.

[18] R. De Vogelaére, *Methods of integration which preserve the contact transformation property of the Hamiltonian equations*, Report No. 4, Dept. of Mathematics, Univ. of Notre Dame, Notre Dame, IN, 1956.

[19] E. J. Doedel, AUTO: *A program for the bifurcation analysis of autonomous systems*, Congr. Numer., 30 (1981), pp. 265–285.

[20] C. Foias, M. S. Jolly, I. G. Kevrekidis, G. R. Sell, and E. S. Titi, *On the computation of inertial manifolds*, Phys. Lett. A, 134 (1988), pp. 433–436.

[21] C. Foias, M. S. Jolly, I. G. Kevrekidis, and E. S. Titi, *Dissipativity of numerical schemes*, Nonlinearity, 4 (1991), pp. 591–613.

[22] C. Foias, O. Manley, and L. Sirovich, *Empirical and Stokes eigenfunctions and the far dissipative turbulent spectrum*, Phys. Fluids A, 2 (1990), pp. 464–467.

[23] C. Foias, O. Manley, and R. Témam, *Modeling of the interaction of small and large eddies in two dimensional turbulent flows*, Math. Model. Numer. Anal., 22 (1988), pp. 93–118.

[24] ———, *Approximate inertial manifold and effective viscosity in turbulent flows*, Phys. Fluids A, 3 (1991), pp. 898–911.

[25] C. Foias, B. Nicolaenko, G. R. Sell, and R. Témam, *Inertial manifolds for the Kuramoto–Sivashinsky equation and an estimate of their lowest dimensions*, J. Math. Pures Appl., 67 (1988), pp. 197–226.

[26] C. Foias, G. R. Sell, and R. Témam, *Inertial manifold for nonlinear evolutionary equations*, J. Differential Equations, 73 (1988), pp. 199–224.

[27] C. Foias, G. R. Sell, and E. S. Titi, *Exponential tracking and approximation of inertial manifolds for dissipative equations*, J. Dynamics Differential Equations, 1 (1989), pp. 199–224.

[28] C. Foias and R. Témam, *Gevrey class regularity for the solutions of the Navier–Stokes equations*, J. Funct. Anal., 87 (1989), pp. 359–369.

[29] C. Foias and E. S. Titi, *Determining nodes, finite difference schemes and inertial manifolds*, Nonlinearity, 4 (1991), pp. 135–153.

[30] A. L. Frenkel, A. J. Babchin, B. G. Levich, T. Schlang, and G. I. Sivashinsky, *Annular flows can keep unstable films from breakup: Nonlinear saturation of capillary instability*, J. Colloid. Interface Sci., 115 (1987), pp. 225–233.

[31] U. Frisch, Z. S. She, and O. Thual, *Viscoelastic behavior of cellular solutions to the Kuramoto–Sivashinsky model*, J. Fluid Mech., 168 (1986), pp. 221–240.

[32] M. N. Glauser, S. J. Leib, and W. K. George, *Coherent structures in the axisymmetric jet mixing layer*, Turbulent Shear Flow, 5, Springer-Verlag, Berlin, New York, 1987.

[33] M. Glauser, X. Zheng, and W. K. George, *The streamwise evolution of coherent structures in the axisymmetric jet mixing layer*, in The Lumley Symposium: Recent Developments in Turbulence, Newport News, VA, Nov. 11–13, 1990.

[34] A. Glezer, Z. Kadioglu, and A. J. Pearlstein, *Development of an extended proper orthogonal decomposition and its application to a time periodically forced plane mixing layer*, Phys. Fluids A, 1 (1989), pp. 1363–1373.

[35] A. P. Hooper and R. Grimshaw, *Nonlinear instability at the interface between two fluids*, Phys. Fluids, 28 (1985), pp. 37–45.

[36] J. M. Hyman, B. Nicolaenko, and S. Zaleski, *Order and complexity in the Kuramoto–Sivashinsky model of weakly turbulent interfaces*, Phys. D, 23 (1986), pp. 265–292.

[37] Ju. Il'Yashenko, *Global analysis of the phase portrait for the Kuramoto–Sivashinsky equation*, IMA Preprint, No. 665, IMA, Univ. of Minnesota, Minneapolis, MN, 1990.

[38] M. S. Jolly, I. G. Kevrekidis, and E. S. Titi, *Approximating inertial manifolds for the Kuramoto–Sivashinsky equation: Analysis and computations*, Phys. D, 44 (1990), pp. 38–60.

[39] ———, *Preserving dissipation in approximate inertial forms for the Kuramoto–Sivashinsky equation*, J. Dynamics Differential Equations, 3 (1991), pp. 179–197.

[40] L. Keefe, *Dynamics of perturbed wavetrain solutions to the Ginzburg–Landau equation*, Stud. Appl. Math., 73 (1985), pp. 91–153.

[41] I. G. Kevrekidis, B. Nicolaenko, and C. Scovel, *Back in the saddle again: a computer assisted study of the Kuramoto–Sivashinsky equation*, SIAM J. Appl. Math., 50 (1990), pp. 760–790.

[42] M. Kirby, D. Armbruster, and W. Guttinger, *An approach for the analysis of spatially localized oscillations*, Proc. Conf. Bifurcation and Chaos, Würzburg, Germany, Aug. 1990.

[43] M. Kirby and D. Armbruster, *Reconstructing phase space for* PDE *simulations*, 1991, preprint.

[44] M. Kwak, *Finite dimensional inertial manifolds for the* 2D *Navier–Stokes equations*, Indiana Univ. Math. J., to appear.

[45] M. Loève, *Probability Theory*, Van Nostrand, New York, 1955.

[46] J. L. Lumley, *The structure of inhomogeneous turbulence*, in Atmospheric Turbulence and Radio Wave Propagation, A. M. Yaglom and V. I. Tatarski, eds., Nauka, Moscow, 1967.

[47] ———, *Stochastic Tools in Turbulence*, Academic Press, New York, 1972.

[48] ———, *Coherent structures in turbulence*, in Transition and Turbulence, R. E. Meyer, ed., Academic Press, New York, 1981, pp. 215–242.

[49] M. Marion, *Approximate inertial manifolds for reaction-diffusion equations in high space dimension*, J. Dynamics Differential Equations, 1 (1989), pp. 245–267.

[50] P. Moin, *Probing turbulence via large eddy simulation*, AIAA 22nd Aerospace Sciences Meeting, Jan. 9–12, 1984, Reno, NV.

[51] P. Moin and R. D. Moser, *Characteristic-eddy decomposition of turbulence in a channel*, J. Fluid Mech., 200 (1989), pp. 471–509.

[52] B. Nicolaenko, B. Scheurer, and R. Témam, *Some global dynamical properties of the Kuramoto–Sivashinsky equation: Nonlinear stability and attractors*, Phys. D, 16 (1985), pp. 155–183.

[53] D. T. Papageorgiou, C. Maldarelli, and D. S. Rumschitsky, *Nonlinear interfacial stability of coreannular film flow*, Phys. Fluids A, 2 (1990), pp. 340–352.

[54] D. T. Papageorgiou and Y. S. Smyrlis, *The route to chaos for the Kuramoto–Sivashinsky equation*, Theoret. Computational Fluid Dynamics, 1991, to appear.

[55] F. R. Payne and J. L. Lumley, *Large eddy structure of the turbulent wake behind a circular cylinder*, Phys. Fluids, (1967), pp. S194–S196.

[56] F. Riesz and B. Sz.-Nagy, *Functional Analysis*, Frederick Ungar Publishing Co., New York, 1955.

[57] J. D. Rodriguez and L. Sirovich, *Low dimensional dynamics for the Ginzburg–Landau equation*, Phys. D, 43 (1990), pp. 77–86.

[58] L. Sirovich, *Turbulence and the dynamics of coherent structures, Parts* I, II, III, Quart. Appl. Math., XLV (1987), pp. 561–590.

[59] L. Sirovich and J.D. Rodriguez, *Coherent structures and chaos: A model problem*, Phys. Lett. A, 120 (1987), pp. 211–214.

[60] L. Sirovich, M. Maxey, and H. Tarman, *Analysis of thermal convection*, in Proc. Sixth Turbulent Shear Flows Symposium, B. E. Launder, ed., Springer-Verlag, New York, 1987.

[61] L. SIROVICH AND H. PARK, *Turbulent convection in a finite domain: Parts* I *and* II, Phys. Fluids A, 2 (1990), pp. 1659–1668.

[62] G. I. SIVASHINSKY, *Nonlinear analysis of hydrodynamic instability in laminar flames, Part* I. *Derivation of basic equations*, Acta Astronautica, 4 (1977), pp. 1176–1206.

[63] ———, *On flame propagation under conditions of stoichiometry*, SIAM J. Appl. Math., 39 (1980), pp. 67–82.

[64] G. I. SIVASHINSKY AND D. M. MICHELSON, *On irregular wavy flow of a liquid down a vertical plane*, Progr. Theoret. Phys., 63 (1980), pp. 2112–2114.

[65] R. TÉMAM, *Infinite Dimensional Dynamical Systems in Mechanics and Physics*, Springer-Verlag, New York, 1989.

[66] ———, *Attractors for the Navier–Stokes equations: localization and approximation*, J. Fac. Sci., Univ. Tokyo, Sec. IA Math., 36 (1989), pp. 629–647.

[67] E. S. TITI, *On approximate inertial manifolds to the Navier–Stokes equations*, J. Math. Anal. Appl., 149 (1990), pp. 540–557.

[68] G. ZHONG AND J. E. MARSDEN, *Lie–Poisson Hamiltonian–Jacobi theory and Lie–Poisson integrators*, Phys. Lett. A, 133 (1988), pp. 134–139.

SIAM J. SCI. COMPUT.
Vol. 14, No. 2, pp. 506–510, March 1993

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific 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.

WAVELETS AND MULTIGRID*

WILLIAM L. BRIGGS† AND VAN EMDEN HENSON‡

Abstract. This note explores the suggestive similarities between wavelet (multiresolution) and multigrid approaches to general operator equations. After presenting the essentials of wavelet and multigrid methods, it can be shown that classical multigrid methods use near-orthogonal basis functions to provide the same orthogonal decomposition of the fine grid space Ω^h that multiresolution methods generate for the maximum resolution space V_0.

Key words. wavelets, multigrid

AMS(MOS) subject classification. 65M55

1. Introducton.

The last few years have seen a remarkable amount of activity and interest in the field of wavelet theory and multiresolution analysis. With this heightened level of interest, researchers in diverse fields have begun to consider wavelet-based methods. The work presented in this paper was done in an exploratory spirit, by investigating the very suggestive similarities between multiresolution analysis and multigrid methods. The results are preliminary and only point to several avenues of future work.

Like many mathematical topics that suddenly gain currency, wavelet theory has origins that are not particularly recent. Both the history and the theoretical foundations of wavelets can be found in several recent and outstanding papers [3], [5], [6], [8], [9], [12]. By all accounts, the definitive treatise is the book by Meyer [11]. In this paper we have neither the space nor the audacity to duplicate the excellent presentations that already exist in these sources. Instead, we will review the essential features of multiresolution analysis that seem to pertain to multigrid algorithms.

2. Multiresolution analysis.

A *multiresolution analysis* is a framework that consists of a sequence of nested closed subspaces (typically of $L^2(\mathbf{R})$),

$$\cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \cdots,$$

whose union is dense in $L^2(\mathbf{R})$. The important features of these subsets are that:

(a) V_0 is spanned by an orthonormal set consisting of integer translations of a single *scaling function* ϕ. For each integer j, V_j is spanned by an orthonormal set consisting of translations of scaled versions of ϕ:

$$V_j = \text{span } \{\phi(2^{-j}x - k)\}_k.$$

(b) For each integer j, $V_j = V_{j+1} \oplus W_{j+1}$, where each W_j is spanned by an orthonormal set consisting of translations of scaled versions of a single *wavelet function* ψ:

*Received by the editors June 10, 1991; accepted for publication September 6, 1992.

†Mathematics Department and Computational Mathematics Group, University of Colorado at Denver, Denver, Colorado 80217–3364.

‡Mathematics Department, Naval Postgraduate School, Monterey, California 93493.

$$W_j = \text{ span } \{\psi(2^{-j}x - k)\}_k.$$

(c) The doubly indexed set $\{\psi(2^{-j}x - k)\}_{j,k}$ spans $L^2(\mathbf{R})$.

Once a scaling function ϕ is found, the associated wavelet function ψ with all of the required orthogonality properties can be found directly. However, ϕ is clearly a rather extraordinary function and the discovery of such functions has been a major research quest. Computationally, there are several ways to produce a scaling function, among them to compute its Fourier transform first. An important property of scaling functions and the associated wavelet functions is that they are highly localized in both the spatial and the frequency domains. Summarizing a wealth of fascinating work, there appear to be three general classes of scaling functions:

(a) C^∞ scaling functions due to Meyer [10] that have noncompact support and polynomial decay,

(b) C^k scaling functions due to Battle [1] and Lemarie [7] that have noncompact support and exponential decay and are generated by orthogonalization of classical splines, and

(c) the Daubechies scaling functions D_{2k} [3] that have compact support, but smoothness that increases slowly with k.

In practice, when dealing with discrete problems such as image processing or signal analysis, the problem is posed on (or projected onto) a finite-dimensional space V_0 which represents the highest level of resolution that is desired. In a multigrid setting V_0 corresponds to Ω^h, the space of fine grid vectors. Given a function $u \in V_0$, the subspaces $V_1, \ldots, V_M$, for some M (corresponding to the coarse grid spaces $\Omega^{2h}, \Omega^{4h}, \ldots$ in multigrid) give representations of u on increasingly coarse levels. At each level, the difference between the projection of u in the spaces V_{j+1} and V_j is given by the projection of u in the space W_{j+1}. The V_j projection retains the smooth features of u, while the W_j projection captures the detail (or oscillatory) components of u.

Within the multiresolution framework it is possible to do a very efficient decomposition of a function over all of the subspaces of interest. Given a function $u \in V_0$, the orthogonality of ϕ and ψ may be used to find coefficients c_{0k} such that

$$u(x) = \sum_k c_{0k}\phi(x-k). \tag{1}$$

This may be regarded as a fine grid representation of u. Furthermore, coefficients c_{1k} and d_{1k} may be found for a coarse grid representation of u on V_1 and W_1 of the form

$$u(x) = \sum_k c_{1k}\phi\left(\frac{x}{2} - k\right) + d_{1k}\psi\left(\frac{x}{2} - k\right). \tag{2}$$

This process may be continued by decomposing each V_j representation of u on the next coarser pair of grids V_{j+1} and W_{j+1} until the coarsest grid is reached. The efficient Pyramid Algorithm for performing this decomposition (and the inverse synthesis) has been proposed by Mallat [8], [9]. The full decomposition of an $N = 2^M$-point sample of u over M levels requires $O(N)$ operations.

3. The multigrid connection.

With this brief survey, we turn to possible connections between multiresolution analysis and classical multigrid algorithms. We will consider a general operator equation of the form $Lu = f$ where L is a self-adjoint operator representing, for example, an elliptic boundary value problem. The notation can be simplified by letting

$$\phi_{jk} = \phi(2^{-j}x - k) \quad \text{and} \quad \psi_{jk} = \psi(2^{-j}x - k).$$

In addition, $\langle u, v\rangle$ will denote the appropriate inner product for the problem.

On the fine grid V_0 and the first coarse grid (V_1, W_1), the solution u may be represented as in (1) and (2). The data f also have a representation on V_0 and (V_1, W_1) with respective coefficients f_{0k}, f_{1k}, and g_{1k}. The fine grid problem after using orthogonality in a standard Galerkin way has the form

$$\sum_k c_{0k}\langle \phi_{0j}, L\phi_{0k}\rangle = f_{0j} \quad \forall j. \tag{3}$$

The known wavelets appear to have no special orthogonality properties with respect to standard elliptic operators so (3) represents a system of linear equations with generally narrow bandwidth, but with no obvious advantages over known discretizations.

In a similar way, the problem may also be represented on the coarse grid. Substituting the (V_1, W_1) representations for u and f and using orthogonality leads to

$$\sum_k c_{1k}\langle \phi_{1j}, L\phi_{1k}\rangle + d_{1k}\langle \phi_{1j}, L\psi_{1k}\rangle = f_{1j} \quad \forall j \tag{4}$$

and

$$\sum_k c_{1k}\langle \psi_{1j}, L\phi_{1k}\rangle + d_{1k}\langle \psi_{1j}, L\psi_{1k}\rangle = g_{1j} \quad \forall j. \tag{5}$$

The problem given in (4) may be regarded as the coarse grid problem for the smooth components of the solution, while (5) gives the coarse grid problem for the oscillatory components.

In classical multigrid algorithms, a solution is sought on a fine grid Ω^h (with grid spacing h) by using standard relaxation methods to approximate errors on the coarse grids $\Omega^{2h}, \Omega^{4h}, \ldots$. The details of the various multigrid cycles are not important for this discussion [2], except to say that vectors are transferred from coarse grids to finer grids with an interpolation operator denoted I_{2h}^{h}, while vectors are transferred from fine grids to coarser grids with a restriction operator denoted I_h^{2h}. The most common choice for interpolation is linear interpolation, while full weighting (averaging of neighbors) is one of the most commonly used restriction operators.

The use of linear interpolation in multigrid schemes corresponds to a representation of the solution in terms of piecewise linear hat functions ϕ_{0k}. The hat functions lack the required orthogonality to be genuine scaling functions. Nevertheless, an associated "wavelet" function ψ may be found for the hat functions (Fig. 1) that allows for an orthogonal decomposition of the fine grid space (V_0 or Ω^h). It is known that the functions ϕ_{1k} span the range of the interpolation operator I_{2h}^{h}, while the functions ψ_{1k} span the nullspace of the full weighting operator I_h^{2h}. Therefore, in multiresolution terms, we would write

$$V_0 = \text{ span } \{\phi_{1k}\} \oplus \text{ span } \{\psi_{1k}\} = V_1 \oplus W_1,$$

while in multigrid terms we would write

$$\Omega^h = \text{ span } \{\phi_{1k}\} \oplus \text{ span } \{\psi_{1k}\} = \text{ Range } \{I_{2h}^{h}\} \oplus \text{ Nullspace } \{I_h^{2h}\}.$$

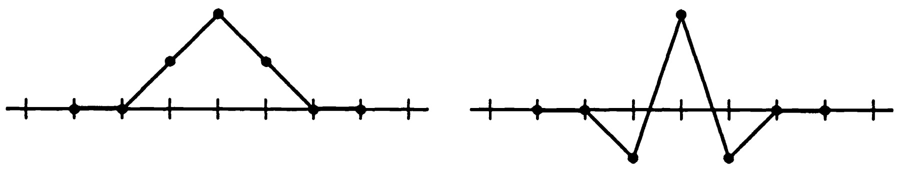


FIG. 1. *The hat functions (left) span V_1 (or the range of interpolation), while the teeth functions (right) span W_1 (or the nullspace of full weighting). These spaces provide an orthogonal decomposition of the fine grid space V_0 (or Ω^h).*

Thus multigrid with piecewise linear interpolation produces the same orthogonal decomposition of the fine grid Ω^h that multiresolution produces of the space V_0.

Classical multigrid methods apply limited relaxation to the fine grid problem (3). They deal only with the coarse grid equation for the smooth components (4). Furthermore, the second term of (4) representing oscillatory components is dropped and the matrix given by $\langle \phi_{1j}, L\phi_{1k} \rangle$ is precisely the multigrid coarse grid operator $L^{2h} = I_h^{2h} L^h I_{2h}^h$. The entire coarse grid equation for the oscillatory components (5) is also dropped in multigrid. The rationale for neglecting the oscillatory components in (4) and (5) is that relaxation is an extremely effective way to isolate or eliminate them.

In summary, multigrid formulations use simple, near-orthogonal basis functions that still allow for an orthogonal decomposition of the fine grid space (and similar decompositions of subsequent coarse grid spaces). Furthermore, multigrid does not attempt to solve for the oscillatory (W_j) components of the solution directly, but rather lets relaxation handle them indirectly. This choice of departing from orthogonality and incorporating relaxation (as well as the residual equation) accounts for the extreme efficiency of multigrid algorithms.

In closing, it should be said that preliminary work on wavelet-based multigrid algorithms for differential equations is in progress [4]. It appears that accuracy comparable to multigrid algorithms can be obtained using the D_{2k} compact wavelets on boundary value problems. However, considerable work on wavelet-based multilevel methods remains to be done.

REFERENCES

[1] G. BATTLE, *A block spin construction of ondelettes, Part 1 : Lemarie functions*, Comm. Math. Phys., 110(1987), pp. 601–615.

[2] W. L. BRIGGS, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[3] I. DAUBECHIES, *Orthonormal tases of compactly supported wavelets*, Comm. Pure Appl. Math., 41(1988), pp. 909–996.

[4] R. GLOWINSKI, W. LAWTON, M. RAVACHOL, AND E. TENEBAUM, *Wavelet solution of linear and nonlinear elliptic, parabolic and hyperbolic problems in one space dimension*, in Computing Methods in Applied Sciences and Engineering, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 55–120.

[5] A. GROSSMAN AND J. MORLET, *Decomposition of Hardy functions into square integrable wavelets of constant shape*, SIAM J. Math. Anal., 15(1984), pp. 723–736.

[6] C. HEIL AND D. WALNUT, *Continuous and discrete wavelet transforms*, SIAM Rev., 31(1989), pp. 628–666.

[7] P. G. LEMARIE, *Ondelettes a localisation exponentielles*, J. Math. Pures Appl., to appear.

[8] S. MALLAT, *Multiresolution approximations and wavelet orthonormal bases of $L^2(\mathbf{R})$*, Trans. Amer. Math. Soc., 315(1989), pp. 69–87.

[9] S. MALLAT, *A theory for multiresolution signal decomposition: The wavelet representation*, IEEE Trans. Pattern Anal. Machine Intell., 11(1989), pp. 674–693.

[10] Y. MEYER, *Ondelettes et fonctions splines, Seminaire equations aux derivees partielles*, Ecole Polytechnique, Paris, 1989.

[11] ———, *Ondelettes et Operateurs*, Hermann, Paris, 1990.

[12] G. STRANG, *Wavelets and dilation equations: A brief introduction*, SIAM Rev., 31(1989), pp. 614–627.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 511-518, May 1993

COMPUTING REPRODUCING KERNELS WITH ARBITRARY BOUNDARY CONSTRAINTS*

C. J. DALZELL† AND J. O. RAMSAY‡

Abstract. Smoothing with L-splines involves the use of the penalty term $\|x\|_2^2 = \int (Lx)^2(t)\,dt$ where L is an arbitrary linear differential operator of order m. It also involves the choice of m constraint functionals to define two orthogonal function subspaces. The basis for a smoothing function is provided by the reproducing kernel for the associated Hilbert space. Computing the kernel for the subspace complementary to $H_1 = \ker L$ can be difficult. Techniques for computing these kernels for arbitrary linear differential operators and constraint functions are developed and illustrated.

Key words. spline smoothing, L-splines, reproducing kernel Hilbert space, Green's function, boundary constraints, nonparametric regression

AMS(MOS) subject classifications. 62J02, 65U05

1. Introduction.

In the spline smoothing problem the objective is to find a function $x(t)$ that will represent the set of observations $(\mathbf{y}, \mathbf{t}) = \{(y_j, t_j), j = 1, \ldots, p\}$. It is assumed that x is an element of the set H^m of functions with $m-1$ absolutely continuous derivatives and a square-integrable mth derivative, $m \geqq 1$. It will also be assumed that x is defined on the real interval $[0, T]$. The definition of the problem usually involves a linear differential operator

$$Lx = \sum_{j=0}^{m} \alpha_j D^j x, \tag{1}$$

where $D^{m-j}\alpha_j$ is an absolutely continuous function of t, $\alpha_m \neq 0$, $D^j x$ denotes the jth derivative of x, and $D^0 x = x$. Let the evaluation operator ρ_t be defined by $\rho_t(x) = (x(t_1), \ldots, x(t_p))'$. Then x is defined to be the minimizer of the smoothing criterion

$$Q(x \mid \mathbf{y}, \lambda) = \|\mathbf{y} - \rho_t(x)\|_N^2 + \lambda \|x\|_2^2, \tag{2}$$

where $\|\mathbf{y}\|_N^2 = \mathbf{y}' N \mathbf{y}$ for some symmetric positive definite matrix N and $\lambda > 0$. The norm in the second term is

$$\|x\|_2^2 = \int_0^T (Lx)^2(t)\,dt. \tag{3}$$

The use of L to define (3) implies that H^m is partitioned into subspace $H_1 = \ker L = \{u \mid Lu = 0\}$ of dimension m and a complementary subspace H_2. The variational problem (2) is solved by defining a reproducing kernel Hilbert space on H^m. This, in turn, involves a choice of inner products and reproducing kernels for the two subspaces H_1 and H_2 with respect to which they are orthogonal. If the bivariate reproducing kernels for H_1 and H_2 are denoted by k_1 and k_2, respectively, then the reproducing kernel for $H^m = H_0 = H_1 \oplus H_2$ is $k_0 = k_1 + k_2$, and these kernels satisfy the equations

$$\langle k_h(s, \cdot), x \rangle_h = x(s) \quad \forall x \in H_h, \quad h = 0, 1, 2.$$

* Received by the editors October 1, 1990; accepted for publication (in revised form) February 17, 1992. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada through grants A320 and 3-31552.

† Canadian Red Cross, 1800 Alta Vista Drive, Ottawa, Ontario, Canada K1G 4J5.

‡ Department of Psychology, 1205 Dr. Penfield Avenue, Montreal, Quebec, Canada H3A 1B1. The work of this author was partially supported by the Educational Testing Service.

Thus the reproducing kernel $k_h(s,\cdot)$ is the representer in Hilbert space H_h of the evaluation functional $\rho_s = x(s)$, the existence of which is assured by the continuity of ρ_s in H^m and the Riesz representation theorem.

The inner products must define orthogonal Hilbert spaces H_1 and H_2, and this can be done by choosing a second linear operator $B: H^m \to R^m$ to be a set of m functionals such that $H_1 = \ker L$ and $H_2 = \ker B$. The orthogonality of H_1 and H_2 implies that $\ker L \cap \ker B = 0$. The inner product for H_1 can then be defined as

$$\langle x, y\rangle_1 = (Bx)'(By). \tag{4}$$

The smoothing function $x \in H_0$ does not depend on the choices of the two inner products and their associated reproducing kernels for the subspaces H_1 and H_2, and if one's sole interest is in x itself, these choices can be governed by mathematical convenience. In particular, Kimeldorf and Wahba [2], [3] and Wahba [7] describe reproducing kernels for the special class of differential operators

$$L = D\frac{1}{a_1}D\frac{1}{a_2}\cdots D\frac{1}{a_m}, \tag{5}$$

where the inner products for H_1 are based on initial value constraints associated with the differential equation $Lx = e$. In practice, almost all applications of spline smoothing have involved the operator $L = D^m$ associated with polynomial smoothing splines and the initial value constraints

$$B_0 = \{(D^j x)(0), j = 0, \dots, m-1\}.$$

There are important reasons for considering the problem of choosing inner products for H_1 and H_2 that are more general. Not all operators L that an application would suggest fall into the class (5). Moreover, the representations of the data in the subspaces H_1 and H_2 can have substantive significance, and interesting constraint operators B defining these subspaces may not correspond to standard initial or boundary value constraints discussed in most references on differential equations. Besse and Ramsay [1] and Ramsay and Dalzell [4] argued that in statistical applications of spline smoothing it is useful to look at the image of x in H_1 and H_2 separately, and also that these images will naturally depend on the nature of the inner products chosen. They further pointed out that inner products based on initial value constraints often do not have much substantive meaning.

The following example illustrates alternative choices of L and B. Suppose that one has data with a tendency to linear trend, and that it is desired to smooth the data in such a way that the linear component u in the smoothing function $x = u + e$ is identified independently of the nonlinear component e. Let $H = H^2[0, T]$ and $L(x) = D^2x$, for which $\ker L = \text{span}\{1, t\}$. If, for some reason, we require that this linear component u fit the data exactly for the first observation y_1, which we can take to be at $t = 0$, then we would use the initial value boundary functional defined by $B(u) = (u(0), (Du)(0))'$. H_1 then contains functions of the form $u(t) = c_1 + c_2 t$, with inner product $\langle u, v\rangle_1 = u(0)v(0) + (Du)(0)(Dv)(0)$, while H_2 contains functions without initial linear trend in the sense that $e(0) = (De)(0) = 0$ for all $e \in H_2$ and $\langle e, f\rangle_2 = \int D^2eD^2f\,dt$.

This approach might be unsuitable if the initial observation y_1 contains substantial observational error, or if there is no particular reason to pass the smoothing function exactly through this or any other observation. In data analysis, it would often appear more natural to use all observations to define the linear trend, and to use them symmetrically in the sense that the indices for t play identical roles in the smoothing process. In this context, a more natural choice for the boundary functionals would be $B(u)=(\int u(t)\,dt, \int tu(t)\,dt)'$. By setting $B(u)=0$ we are still defining H_1 as span $\{1, t\}$, but now with the inner product

$$\langle u, v\rangle_1 = \left[\int u(t)\,dt\right]\left[\int v(t)\,dt\right] + \left[\int tu(t)\,dt\right]\left[\int tv(t)\,dt\right]. \tag{6}$$

This inner product is uniform in that it depends on the behavior of u and v at all values of t rather than just on $t=0$. H_2 will have the same inner product as in the previous example, but will now consist of functions with linear trend eliminated by averaging across all t values.

Ramsay and Dalzell [4] considered an example in which the smoothing was by the shifted harmonic splines defined by

$$Lx = \omega^2 D + D^3,$$

so that $H_1 = \ker L$ is spanned by $\{1, \sin(\omega t), \cos(\omega t)\}$. In their application, H_2 was defined to be the set of $x \in H^m$ such that

$$\int_0^T x(t)\,dt = 0,$$

$$\int_0^T \sin(\omega t)x(t)\,dt = 0,$$

$$\int_0^T \cos(\omega t)x(t)\,dt = 0.$$

These constraints were natural with the periodic character of their data, consisting of observations of mean monthly temperature and precipitation, since there was no particular reason to single out any one month as the origin for the series.

The reproducing kernel k_1 is easily computed. Let $u_1, \dots, u_m$ be functions spanning H_1 and let $\mathbf{u} = (u_1, \dots, u_m)'$. Defining the matrix U_1 to be the order m symmetric matrix with values $\langle u_i, u_j\rangle_1$, $i, j, = 1, \dots, m$, we have

$$k_1(s, t) = \mathbf{u}(s)' U_1^{-1} \mathbf{u}(t). \tag{7}$$

Thus, for inner product (6), we have that

$$k_1(s, t) = 4(13 - 24(s+t) + 45st).$$

Computing the kernel k_2 is generally much more difficult, and this is the problem principally addressed by this paper. In the following section, the Green's function for the initial value operator B_0 is given, and it is shown how this can be transformed to give the Green's function for any linear operator satisfying the orthogonality condition $\ker L \cap \ker B = 0$. Because the actual reproducing kernel can be very complex if written out completely in univariate algebraic notation, the following section also develops an expression for the Green's function and the reproducing kernel in matrix notation that is more suitable for computation.

2. Computing the reproducing kernel k_2. We proceed here by first computing the Green's function $g_0(t; w)$ associated with the initial value condition

$$B_0 x = (x(0), (Dx)(0), \dots, (D^{m-1}x)(0))' = 0,$$

and then show how to modify this Green's function to obtain the Green's function for the desired B.

In the particular case where e is a real-valued function and L a linear differential operator of order m, the Green's function $g(t;\, w)$ associated with L and the boundary constraints $B(e)=0$ satisfies

$$e(t)=\int g(t;\, w)(Le)(w)\, dw, \quad \forall e \mid B(e)=0, \tag{8}$$

and has the additional properties

$$Bg(\cdot\,; w) \quad \forall w=0,$$
$$(L^*g)(t;\, w)=0 \quad \forall t\neq w,$$

L^* being the adjoint operator associated with L,

$$L^*x=\sum_{j=0}^{m}(-1)^jD^j(\alpha_jx), \tag{9}$$

and applied to g as a function of w (see Roach [5] and Schumaker [6, p. 426]). The Green's function

- provides a convenient means of computing the reproducing kernel,
- is useful when computing L_x,
- links the theory of reproducing kernel Hilbert spaces with the theory of ordinary differential equations.

The Green's function can be derived simply from k_2 since it follows from (8) that $Lk_2(s,\cdot)=g(s;\,\cdot)$, where L is applied to k_2 as a function of w for fixed s. Conversely, the reproducing kernel k_2 can also be computed from g as

$$\begin{aligned} k_2(s,t)&=\langle k_2(s,\cdot), k_2(t,\cdot)\rangle_2\\ &=\int Lk_2(s,w), Lk_2(t,w)\, dw \qquad (10)\\ &=\int g(s;\, w), g(t;\, w)\, dw. \end{aligned}$$

2.1. Green's function for the initial value condition. We follow here the exposition of Schumaker [6, pp. 422–429], in which it is assumed without loss of generality that L is in the form

$$L=D^m+\sum_{j=0}^{m-1}\alpha_jD^jx.$$

Let

$$\mathbf{u}=(u_1,\ldots,u_m)'=\operatorname{span}\{\ker L\}.$$

The Wronskian matrix $\mathbf{W}: T\to R^{mm}$ is the order m matrix-valued function with elements $D^{j-1}u_i$, $i,j=1,\ldots,m$. It is assumed that $\mathbf{W}(w)$ is nonsingular for all w. The adjoint functions $\mathbf{u}^*=(u_1^*,\ldots,u_m^*)'$ are defined by $\mathbf{u}^*=(\mathbf{W}^{-1})_m$, where $(\mathbf{W}^{-1})'_m$ is the mth row of $\mathbf{W}^{-1}$. The Green's function for the initial value condition is (Schumaker [6, Thm. 10.11])

$$g_0(t;\, w)=\mathbf{u}(t)'\mathbf{u}^*(w)=\sum_i^m u_i(t)u_i^*(w), \qquad w\leqq t, \tag{11}$$

and 0 otherwise.

Example 1: *Linear trend.* When $L = D^2$ with $\mathbf{u}(t) = (1, t)'$, $t \in [0, 1]$, we have that

$$W(w) = \begin{bmatrix} 1 & 0 \\ w & 1 \end{bmatrix}$$

and

$$\mathbf{W}(w)^{-1} = \begin{bmatrix} 1 & 0 \\ -w & 1 \end{bmatrix} \tag{12}$$

with $|\mathbf{W}| = 1$. Thus $g_0(t;\, w) = t - w$, $w \leqq t$, and 0 otherwise.

Example 2: *Harmonic variation with mean zero.* When

$$L = \theta^2 I + D^2 \quad \text{with } \mathbf{u}(t) = (\sin(\theta t), \cos(\theta t))',$$

we have that

$$\mathbf{W}(w) = \begin{bmatrix} \sin(\theta w) & \theta \cos(\theta w) \\ \cos(\theta w) & -\theta \sin(\theta w) \end{bmatrix}$$

and

$$\mathbf{W}(t)^{-1} = \begin{bmatrix} \sin(\theta w) & \cos(\theta w) \\ \theta^{-1} \cos(\theta w) & -\theta^{-1} \sin(\theta w) \end{bmatrix} \tag{13}$$

with $|\mathbf{W}| = -\theta^{-1}$. Thus

$$g_0(t;\, w) = \theta^{-1} \sin[\theta(t - w)], \qquad w \leqq t, \tag{14}$$

and 0 otherwise.

Example 3: *Harmonic variation with arbitrary mean.* When

$$L = \theta^2 D + D^3 \quad \text{with } \mathbf{u}(t) = (1, \sin(\theta t), \cos(\theta t))',$$

we have that

$$W(w) = \begin{bmatrix} 1 & 0 & 0 \\ \sin(\theta w) & \theta \cos(\theta w) & -\theta^2 \sin(\theta w) \\ \cos(\theta w) & -\theta \sin(\theta w) & -\theta^2 \cos(\theta w) \end{bmatrix}$$

and

$$\mathbf{W}(t)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \theta^{-1} \cos(\theta w) & -\theta^{-1} \sin(\theta w) \\ \theta^{-2} & -\theta^{-2} \sin(\theta w) & -\theta^{-2} \cos(\theta w) \end{bmatrix} \tag{15}$$

with $|\mathbf{W}| = -\theta^{-3}$. Thus

$$g_0(t;\, w) = \theta^{-2}\{1 - \cos[\theta(t - w)]\}, \qquad w \leqq t, \tag{16}$$

and 0 otherwise.

2.2. Modifying g_0 to obtain g for arbitrary $Bx = 0$. Now let B be an arbitrary set of functions for which $\ker L \cap \ker B = 0$ and define the vector-valued function $\mathbf{b}_0$ by

$$\mathbf{b}_0(w) = Bg_0(\cdot;\, w).$$

Let order m matrix B^* have elements $b_i u_j$, where b_i is the ith functional in B; that is, $B^* = B\mathbf{u}'$.

THEOREM 1. *If B^* is nonsingular and the functionals b_i commute with integration in H^m, then the Green's function for the condition $Bf = 0$ is*

$$g(t;\, w) = g_0(t;\, w) - \mathbf{u}(t)' B^{*^{-1}} \mathbf{b}_0(w). \tag{17}$$

Proof. For any $f_0 \in \ker B_0$ let $f(t) = f_0(t) - \mathbf{u}(t)' B^{*^{-1}}(Bf_0)$. Then $Bf = 0$, and it is easy to show that the mapping $\ker B_0 \to \ker B$ defined in this way is a bijection. By $Lu = 0$,

$$\begin{aligned}\int g_0(t;\, w)(Lf)(w)\, dw &= f_0(t) - [B^{*^{-1}}(Bf_0)]' \int g_0(t;\, w)(Lu)(w)\, dw \\ &= f_0(t).\end{aligned}$$

Consequently,

$$\begin{aligned}\int g(t;\, w)(Lf)(w)\, dw = \int & g_0(t;\, w)(Lf_0)(w)\, dw \\ &- \mathbf{u}(t)' B^{*^{-1}} \int Bg(\cdot\,;\, w)(Lf)(w)\, dw \\ &= f_0(t) - \mathbf{u}(t)' B^{*^{-1}} B \int g(\cdot\,;\, w)(Lf)(w)\, dw \\ &= f_0(t) - u(t)' B^{*^{-1}} B f_0 \\ &= f(t). \qquad \square\end{aligned}$$

The condition of commutability of B and integration will hold for bounded functionals and for evaluations of derivatives up to order $m-1$, which includes all the cases discussed in this paper.

Example 1: *Linear trend with integral constraints.* In this case

$$B^* = (1/6)\begin{bmatrix} 6 & 3 \\ 3 & 2 \end{bmatrix}$$

and

$$\mathbf{b}_0(w) = (1/6)\begin{bmatrix} 3 - 6w + 3w^2 \\ 2 - 3w + w^3 \end{bmatrix},$$

so that

$$g(t;\, w) = \begin{cases} w^3(1-2t) + w^2(3t-2), & w \leqq t, \\ w^3(1-2t) + w^2(3t-2) + w - t, & w > t. \end{cases} \tag{18}$$

This yields

$$\begin{aligned} k_2(s, t) = [&4 - 22(s+t) + 156st - 210st^2 + 70t^3 - 70(t^4 + s^4) \\ &+ 105st(s^3 + t^3) + 21(s^5 + t^5) - 42st(s^4 + t^4)]/420, \qquad s \leqq t. \end{aligned} \tag{19}$$

The symmetry of k_2 provides the value of $k_2(s, t)$, $s > t$.

3. Matrix computation of g and k_2. The integrations involved in transforming g_0 to g as well as the final computation of the reproducing kernel k can be tedious, and are greatly facilitated by symbolic computation software. In particular, computation of k requires that the integration be performed separately for the subintervals $[0, t]$, $[t, s]$, and $[s, T]$, since g will have an expression for $[0, s]$ which differs from that for $[s, T]$. However, the calculations can be simplified as follows.

By (11) and (17), $g(\cdot\,; w) \in \ker L$ for $t \neq w$. Moreover, $g_0(t; \cdot) \in \ker L^*L$ for $t \neq w$, where L^* is the adjoint operator for L. Consequently,

$$g_0(t; w) = \mathbf{v}(w)'C_0\mathbf{u}(t), \qquad w < t,$$

and 0 otherwise, where $\mathbf{v} = (v_1, \cdots, v_{2m})'$ spans $\ker L^*L$. Moreover, $\mathbf{u}(t)'B^{*-1}\mathbf{b}_0(w)$ can be expressed in the form $\mathbf{v}(w)'C_1\mathbf{u}(t)$. Thus

$$g(t; w) = \begin{bmatrix} \mathbf{v}(w)'(C_0 - C_1)\mathbf{u}(t), & w \leqq t \\ -\mathbf{v}(w)'C_1\mathbf{u}(t), & w > t \end{bmatrix}, \tag{20}$$

and finally,

$$k_2(s, t) = \mathbf{u}(s)'[C_0'\mathbf{F}(s)C_0 - C_0'\mathbf{F}(s)C_1 - C_1'\mathbf{F}(t)C_0 + C_1'\mathbf{F}(T)C_1]\mathbf{u}(t), \quad s \leqq t, \tag{21}$$

and $k(s, t) = k(t, s)$, $s > t$, where

$$\mathbf{F}(s) = \int_0^s \mathbf{v}(w)\mathbf{v}(w)'\,dw.$$

These results imply that once the constant matrices C_0 and C_1 and the matrix function $\mathbf{F}$ are computed, the values of k_2 can be computed by simple matrix computations.

Example 2: *Harmonic variation, mean zero with integral constraints.* In this case,

$$v = (\sin(\theta w), \cos(\theta w), w \sin(\theta w), w \sin(\theta w))',$$

so that

$$C_0' = \theta^{-1}\begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}. \tag{22}$$

Matrix $\theta[\theta^2T^2 - \sin^2(\theta T)]C_1$ works out to be

$$\begin{bmatrix} \theta T + \cos(\theta T)\sin(\theta T) & -\theta^2T^2 \\ \theta^2T^2 - \sin^2(\theta T) & 0 \\ -\theta\sin^2(\theta T) & \theta^2T - \theta\cos(\theta T)\sin(\theta T) \\ -\theta^2T - \theta\cos(\theta T)\sin(\theta T) & \theta\sin^2(\theta T) \end{bmatrix}. \tag{23}$$

Example 3: *Harmonic variation with arbitrary mean and integral constraints.* In this case,

$$\mathbf{v} = (1, w, \sin(\theta w), \cos(\theta w), w\sin(\theta w), w\sin(\theta w))',$$

so that

$$C_0' = \theta^{-2}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}. \tag{24}$$

Matrix C_1 can be expressed as C^*B^{*-1} where the matrix $(2\theta)B^*$ is

$$\begin{bmatrix} 2\theta T & 2(1-\cos(\theta T)) & 2\sin(\theta T) \\ 2(1-\cos(\theta T)) & \theta T - \sin(\theta T)\cos(\theta T) & \sin^2(\theta T) \\ 2\sin(\theta T), & \sin^2(\theta T) & \theta T + \sin(\theta T)\cos(\theta T) \end{bmatrix}, \tag{25}$$

and the matrix $(2\theta)^3 C^{*\prime}$ is

$$\begin{bmatrix} 2\theta T & -2\cos(\theta T) & 2\sin(\theta T) \\ -2\theta & 0 & 0 \\ 2\cos(\theta T) & \sin(\theta T)\cos(\theta T)-\theta T & \cos^2(\theta T)-2 \\ -2\sin(\theta T) & 1+\cos^2(\theta T) & -\sin(\theta T)\cos(\theta T)-\theta T \\ 0 & \theta & 0 \\ 0 & 0 & \theta \end{bmatrix}. \tag{26}$$

REFERENCES

[1] P. BESSE AND J. O. RAMSAY, *Principal components analysis of sampled functions*, Psychometrika, 51 (1986), pp. 285-311.

[2] G. S. KIMELDORF AND G. WAHBA, *A correspondence between Bayesian estimation on stochastic processes and smoothing by splines*, Ann. Math. Statist., 41 (1970), pp. 495-502.

[3] ———, *Some results on Tchebycheffian spline functions*, J. Math. Anal. Appl., 33 (1971), pp. 82-94.

[4] J. O. RAMSAY AND C. J. DALZELL, *Some tools for functional data analysis*, J. Roy. Statist. Soc., 53 (1991), pp. 539-572.

[5] G. F. ROACH, *Green's Functions*, Cambridge University Press, London, U.K., 1982.

[6] L. SCHUMAKER, *Spline Functions: Basic Theory*, John Wiley, New York, 1981.

[7] G. WAHBA, *Spline Models for Observational Data*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 519-530, May 1993

SPARSE MATRIX COMPUTATIONS ON PARALLEL PROCESSOR ARRAYS*

ANDREW T. OGIELSKI† AND WILLIAM AIELLO†

Abstract. This paper investigates the balancing of distributed compressed storage of large sparse matrices on a massively parallel computer. For fast computation of matrix-vector and matrix-matrix products on a rectangular processor array with efficient communications along its rows and columns it is required that the nonzero elements of each matrix row or column be distributed among the processors located within the same array row or column, respectively. Randomized packing algorithms are constructed with such properties, and it is proved that with high probability the algorithms produce well-balanced storage for sufficiently large matrices with bounded number of nonzeros in each row and column, but no other restrictions on structure. Then basic matrix-vector multiplication routines are described with fully parallel interprocessor communications and intraprocessor gather and scatter operations. Their efficiency is demonstrated on the 16,384-processor MasPar computer.

Key words. distributed data structures, linear algebra, load balancing, parallel algorithms, randomized algorithms, sparse matrices

AMS(MOS) subject classifications. 65Y05, 65F50, 68P05, 68Q22

1. Introduction.

Efficient computation in the data-parallel mode is achieved with distributed data structures that balance the processors' computation load and promote maximum parallelism of data communications. For parallel dense matrix linear algebra routines [2]-[4], [6], [7], [9], [12], a balanced distribution of computations to processors can be achieved together with very regular interprocessor communication patterns. In contrast, in the design of algorithms for unstructured sparse matrices, the data compression scheme may conflict with efficient communication. For instance, if a large matrix is evenly partitioned into blocks in correspondence with the processor array, the communications are similar to the dense case, but the computation time will be determined by the block with the largest number of nonzeros, which may be unacceptably large. Alternatively, if the nonzeros are densely packed into the processor array without regard for the row and column structure, the benefit of even processor load may be negated by overwhelming communication costs.

Universal distributed data structures that guarantee both fast computation and efficient compression for all sparse matrices have not been found. Therefore, we associate classes of sparse matrices with appropriate data structures and computational kernels. For unstructured sparse matrices, it is desirable to have few classes defined by simple and easily verified properties. Here we will only request that no row or column of a sparse matrix be too dense, with no other restrictions on matrix structure. We assume that the matrix remains unchanged during computation, and that it is accessed sufficiently often to justify preprocessing for well-balanced storage; this is the case in the family of Lanczos algorithms for the symmetric eigenvalue problem, conjugate gradient methods, and in many other numerical algorithms [7].

Suppose we have an $m \times n$ two-dimensional processor array with efficient communications along its rows and columns. Many parallel computers can be configured in this way. The processing element in the ith row and the jth column will be called

* Received by the editors August 19, 1991; accepted for publication (in revised form) May 11, 1992.
† Bell Communications Research, Morristown, New Jersey 07960.

$PE(i,j)$. Let A be a sparse $M \times N$ matrix, $M \gg m$ and $N \gg n$. We will say that an assignment of matrix elements to the processor array preserves the *integrity* of the matrix if for every row (column) all its nonzero elements are placed into processors lying in a single row (column) of the array. In such an assignment each processor stores a submatrix, and the data communications required by linear operations can be carried out in parallel. The problem of finding an assignment that preserves the integrity of the matrix and minimizes the largest processor load is NP-complete. This can easily be shown by a reduction from bin packing [5]. Nonetheless, we show that if we do not require a deterministic, optimal solution, then a fast random assignment that preserves the matrix integrity and comes very close to optimal can be achieved with high probability.

We analyze two schemes for random assignments that preserve the integrity. In the simplest random assignment scheme each matrix row index is randomly and independently assigned a row of the processor array, and each matrix column index is randomly and independently assigned a column of the array. The dimensions of submatrices stored in different processors generally will be different.

This loss of uniformity is corrected in the second, more restricted scheme. Suppose that before the loading a random permutation of matrix rows and a random permutation of matrix columns are performed, and then the permuted matrix is partitioned into blocks of size $\lceil M/m \rceil \times \lceil N/n \rceil$ (the right-most and lowest blocks may be smaller). This results in an $m \times n$ matrix of blocks which are assigned in the natural way to the $m \times n$ processor array.

The second scheme is very convenient for algorithm design, but a priori it is not obvious that it can achieve as good a load balancing as the first method. Suppose that the matrix A has T nonzero elements and at most $R(C)$ nonzeros in each row (column). Let $|P_{i,j}|$ denote the number of nonzeros assigned to processor $PE(i,j)$; it is easily seen that for any assignment scheme we must have $\max_{i,j} |P_{i,j}| \geqq \lceil T/mn \rceil$. We prove that for sufficiently large T and sufficiently small R and C, either assignment scheme produces a well-balanced load with high probability. This is stated by showing that in both schemes $\Pr\{\max_{i,j} |P_{i,j}| \geqq (1+\varepsilon)^2 \lceil T/mn \rceil\}$ is bounded from above by $\exp\{-O(\varepsilon h(\varepsilon))\}$, where $h(x) = (1+x^{-1}) \ln(1+x) - 1$. The function $h(x)$ is strictly monotonically increasing, and $h(x) \approx x/2$ for $x \to 0$, $h(x) \approx \ln(x)$ for $x \to \infty$.

As far as we know, this is the first result on parallel sparse matrix computations with provably good storage efficiency for unstructured sparse matrices. The exact statement of the theorem and its proof are given in § 2. The proof for the first scheme makes repeated use of Bennett's inequality for large deviations from the expected value of a sum of independent random variables [1]. The second scheme uses an extension of this inequality to a special case of dependent random variables due to Hoeffding [8]. Incidently, very large randomly sparse matrices (where each element independently is nonzero with a fixed probability) with high probability give balanced load in the second scheme even without the row and column permutations (apply Bennett's inequality directly to block submatrices).

Once submatrices are assigned to processor elements, the nonzeros are stored in a compressed format. Although the structure (i.e., location of nonzeros) of submatrices stored in different processing elements (PEs) may vary, this does not impede the parallelism of data movement and execution of matrix primitives if scatter and gather techniques are used: data can be transmitted in parallel in regular patterns to buffers in PEs and then scattered to proper memory locations, or vice versa. For this we assume that the parallel computer supports indirect addressing; that is, the PEs can store pointers to their own memories. With indirect addressing the scatter and gather operations can be executed in parallel even on data-parallel machines, since each PE can access a different memory location in a single instruction.

In § 3 we assume a balanced distribution of nonzeros and show one possible implementation of the basic matrix-vector kernels for computation of Ax or $y^T A$ and their extensions to blocks of vectors assuming only the data-parallel computer model. We also present examples of both the efficiency of packing and the performance of kernels on a particular data-parallel computer, the 16,384-processor MasPar MP-1216.

The NP-completeness of the optimum integrity-preserving matrix assignment problem does not preclude the possibility that a deterministic, polynomial time algorithm may be able to produce an assignment that provably comes within a constant factor of the optimum. Finding such an algorithm is an interesting question for future research. We also expect that randomized storage of data in distributed data structures preserving the favorable communication patterns will be useful in other applications.

2. Balanced loading.

To distinguish the matrix row and column indices from the processor array indices we always use capital letters $I, J, \ldots$ for the matrix, and lower case letters $i, j, \ldots$ for the processors. For brevity, in this section we use $P_{i,j}$ to denote the processor $PE(i, j)$. Also, we will denote the set of integers from a to b inclusive by $[a, b]$.

We consider assignments of nonzero elements of an $M \times N$ sparse matrix A to processors in an $m \times n$ array (where $M \gg m$ and $N \gg n$) which preserve the integrity of matrix rows and columns. Any such assignment, by definition, can be described by two mappings: The row mapping $\rho : [0, M-1] \rightarrow [0, m-1]$, and the column mapping $\sigma : [0, N-1] \rightarrow [0, n-1]$. A matrix element $A_{I,J}$ is assigned to $P_{\rho(I),\sigma(J)}$; therefore, the processor P_{ij} stores the submatrix of A given by the rows in $\rho^{-1}(i)$ and the columns in $\sigma^{-1}(j)$. Let $|P_{i,j}|$ denote the number of nonzeros in this submatrix.

We analyze two fast probabilistic algorithms for the construction of row and column mappings that attempt to keep all the $|P_{i,j}|$ as close to the optimum value $\lceil T/mn \rceil$ as possible. To describe the first, let $\mathcal{F}_{M,m}$ be the uniform distribution on all functions from $[0, M-1]$ to $[0, m-1]$. We draw ρ according to $\mathcal{F}_{M,m}$ and σ according to $\mathcal{F}_{N,n}$. An explicit method to construct such random mappings is to assign a uniform random value from $[0, m-1]$ to $\rho(I)$ independently for each $I \in [0, M-1]$, and to assign a uniform random value form $[0, n-1]$ to $\sigma(J)$ independently for all $J \in [0, N-1]$.

The second type of random mapping is as follows. Let $\mathcal{G}_{M,m}$ be the uniform distribution on all functions ρ from $[0, M-1]$ to $[0, m-1]$ such that $|\rho^{-1}(i)| = \lceil M/m \rceil$ for $i \in [0, (M \bmod m) - 1]$ and $|\rho^{-1}(i)| = \lfloor M/m \rfloor$ for $i \in [M \bmod m, m-1]$. The row mapping is drawn according to $\mathcal{G}_{M,m}$ and the column mapping is drawn according to $\mathcal{G}_{N,n}$. This leads to the assignment scheme mentioned in the Introduction, since one way to generate a mapping according to $\mathcal{G}_{M,m}$ is to take a random permutation π of $[0, M-1]$ and break the permuted sequence into $M \bmod m$ consecutive subsequences of length $\lceil M/m \rceil$ and $m - (M \bmod m)$ subsequences of length $\lfloor M/m \rfloor$. Alternatively, we may use $\rho(I) = \pi(I) \bmod m$. This restriction on the size of each submatrix actually forces a slight inefficiency, as we will see. For ease of notation in handling this, let m' and n' be $M/\lceil M/m \rceil$ and $N/\lceil N/n \rceil$, respectively, when we are dealing with the distributions $\mathcal{G}_{M,m}$ and $\mathcal{G}_{N,n}$, and let them be simply m and n when dealing with the distributions $\mathcal{F}_{M,m}$ and $\mathcal{F}_{N,n}$.

We can now state our main theorem. As defined before, T is the total number of nonzeros of A, R is the largest number of nonzeros in a row, and C is the largest number of nonzeros in a column.

THEOREM. *If the row and column assignments are chosen either according to $\mathcal{F}_{M,m}$ and $\mathcal{F}_{N,n}$, respectively, or according to $\mathcal{G}_{M,m}$ and $\mathcal{G}_{N,n}$, respectively, then for any $\varepsilon > 0$ the*

$$\Pr\left\{\max_{ij}\{|P_{i,j}|\} \geqq (1+\varepsilon)^2 \frac{T}{m'n'}\right\}$$

is at most the minimum of

$$mn \exp\left\{-\frac{T}{m'R}\,\varepsilon h(\varepsilon)\right\} + Mn \exp\left\{-\frac{R}{n'}\,\varepsilon h(\varepsilon)\right\} + n \exp\left\{-\frac{T}{n'C}\,\varepsilon h(\varepsilon)\right\}$$

and

$$mn \exp\left\{-\frac{T}{n'C}\,\varepsilon h(\varepsilon)\right\} + Nm \exp\left\{-\frac{C}{m'}\,\varepsilon h(\varepsilon)\right\} + m \exp\left\{-\frac{T}{m'R}\,\varepsilon h(\varepsilon)\right\},$$

where $h(x) = (1+x^{-1})\ln(1+x) - 1$.

Proof. We first consider the case when the pair of index mappings (ρ, σ) is drawn according to $\mathcal{F}_{M,m} \times \mathcal{F}_{N,n}$. The proof involves three applications of a large deviation theorem due to Bennett [1]. When (ρ, σ) is drawn according to $\mathcal{G}_{M,m} \times \mathcal{G}_{N,n}$, the proof follows immediately once we argue that a certain extension of Bennett's inequality due to Hoeffding [8] applies.

For the statement of Bennett's inequality let $\xi_0, \xi_1, \ldots, \xi_{M-1}$ be independent random variables, bounded from above, and with finite first and second moments. Let $S = \sum_{I=0}^{M-1} \xi_I$, and let $\mathrm{Var}(S)$ be its variance.

Bennett's inequality. For all $d \geqq 0$ such that $\xi_I - \mathrm{E}(\xi_I) \leqq d$ for all I, and for all $\gamma \geqq 0$,

$$\Pr\{S \geqq \mathrm{E}(S) + \gamma\} \leqq \exp\left\{-\frac{\gamma}{d}\, h\left(\frac{d\gamma}{\mathrm{Var}(S)}\right)\right\}.$$

We need an application of Bennett's inequality to the following. Let $\bar{W} = \{W_0, \ldots, W_{M-1}\}$ be a set of M objects, each having a well-defined nonnegative "weight" $|W_I|$. Define the weight of a subset $\bar{X} \subseteq \bar{W}$ as the sum of the weights of the elements of $\bar{X}$. Suppose that we put each $W_I \in \bar{W}$ independently and uniformly into one of m bins, $\{B_0, B_1, \ldots, B_{m-1}\}$. Formally, each bin B_i, $i \in [0, m-1]$, is the set of objects W_I such that $\rho(I) = i$ where ρ is drawn according to $\mathcal{F}_{M,m}$. The weight of each bin $|B_i|$ is the sum of the weights of the elements in the bin. This can be written as $|B_i^{\bar{W}}| = \sum_{I=0}^{M-1}(\rho(I)=i)|W_I|$, where $(\rho(I)=i)$ is the indicator variable which is one when $\rho(I)=i$ and zero otherwise. Since ρ is drawn according to $\mathcal{F}_{M,m}$, for any fixed bin $i \in [0, m-1]$, the 0-1 random variables $(\rho(0)=i)$, $(\rho(1)=i), \ldots, (\rho(M-1)=i)$ are independent Bernoulli variables with probability of success $1/m$. Hence, the weight of each bin $|B_i^{\bar{W}}|$ has the same distribution as the weighted sum of independently and identically distributed Bernoulli variables (although $|B_0^{\bar{W}}|, \ldots, |B_{m-1}^{\bar{W}}|$ are themselves dependent random variables).

If we know a $w > 0$ such that $0 \leqq |W_I| \leqq w$ for all $I \in [0, M-1]$ then the expected value $\mathrm{E}(|B_i^{\bar{W}}|) = (1/m)\sum_{I=0}^{M-1}|W_I| = |\bar{W}|/m$, and variance,

$$\mathrm{Var}(|B_i^{\bar{W}}|) = \sum_{I=0}^{M-1} \mathrm{Var}((\rho(I)=i)|W_I|) = (1/m)(1-1/m)\sum_{I=0}^{M-1}|W_I|^2,$$

are bounded. Furthermore, $(\rho(I)=i)|W_I| - E((\rho(I)=i)|W_I|) \leqq (1-1/m)|W_I| \leqq (1-1/m)w$ for all $I \in [0, M-1]$. Hence, we may apply Bennett's inequality with

$d=(1-1/m)w$ and the union bound to get a bound on the weight of the largest bin,

$$\Pr\left\{\max_{i\in[0,m-1]}\{|B_i^{\bar{W}}|\}\geqq\frac{|\bar{W}|}{m}+\gamma\right\}\leqq m\exp\left\{-\frac{\gamma}{(1-1/m)w}h\left(\frac{wm\gamma}{\sum_I|W_I|^2}\right)\right\}$$
$$\leqq m\exp\left\{-\frac{\gamma}{w}h\left(\frac{wm\gamma}{\sum_I|W_I|^2}\right)\right\}.$$

Unfortunately, in the course of the remainder of the proof, we may not know enough about $\bar{W}$ to compute $\sum_{I=0}^{M-1}|W_I|$ or $\sum_{I=0}^{M-1}|W_I|^2$ exactly. Nonetheless, we will know an upper bound $W>0$ on the sum of the weights, $\sum_{I=0}^{M-1}|W_I|\leqq W$. This implies an upper bound of wW on $\sum_{I=0}^{M-1}|W_I|^2$. To see this, note that $\sum_{I=0}^{M-1}|W_I|^2\leqq w\sum_{I=0}^{M-1}|W_I|$ since $0\leqq|W_I|\leqq w$. This, in turn, is at most wW by the bound on the sum of the weights. Now we can get a simple upper bound on the probability that one of the m bins has large weight

$$(*)\quad \Pr\left\{\max_{i\in[0,m-1]}\{|B_i^{\bar{W}}|\}\geqq(1+\varepsilon)\frac{W}{m}\right\}\leqq\Pr\left\{\max_{i\in[0,m-1]}\{|B_i^{\bar{W}}|\}\geqq\frac{|\bar{W}|}{m}+\varepsilon\frac{W}{m}\right\}$$
$$\leqq m\exp\left\{-\frac{W}{mw}\varepsilon h(\varepsilon)\right\}.$$

The last inequality follows due to the fact that h is monotone.

We return now to the problem of assigning submatrices of A to processors in the array. Let the weight of an element $A_{I,J}$, denoted $|A_{I,J}|$, be 1 if $A_{I,J}$ is a nonzero, and zero otherwise. Let $A_{I,*}$ be the Ith row of A and let $|A_{I,*}|$ be the number of ones in the Ith row. Let R be an upper bound on the weights of the rows of A. Similarly, let $A_{*,J}$ be the Jth column of A and $|A_{*,J}|$ the number of ones in the Jth column. Let C be an upper bound on the weights of the columns of A.

For the purposes of this analysis we make the column assignment, σ, first and then the row assignment, ρ. Let $V_{I,j}$, $I\in[0,M-1]$, and $j\in[0,n-1]$ be the submatrices formed by the column assignment. That is, $V_{I,j}=\{A_{I,J}\,|\,J\in\sigma^{-1}(j)\}$. This defines the matrix V. Let $V_{*,j}$ be the jth column of V. It is the submatrix of A consisting of all the columns, $A_{*,J}$, such that $\sigma(J)=j$. That is, the column assignment σ assigns the objects $\bar{W}=\{A_{*,J}\,|\,J\in[0,N-1]\}$ uniformly and randomly to the bins $\{V_{*,j}\,|\,j\in[0,n-1]\}$. We can apply $(*)$ to bound the weight of the largest bin by recalling that $A_{*,J}$ has weight at most C, since it is simply a column of A, and $|\bar{W}|=\sum_J|A_{*,J}|$ is simply the total number of nonzeros of A, which is at most T. Hence,

$$\Pr_{\mathcal{F}_{N,n}}\left\{\max_{j\in[0,n-1]}\{|V_{*,j}|\}\geqq\frac{T}{n}(1+\varepsilon_3)\right\}\leqq n\exp\left\{-\frac{T}{nC}\varepsilon_3h(\varepsilon_3)\right\}.$$

A similar analysis can also be applied on a row by row basis. For a fixed row $I\in[0,M-1]$, σ assigns the Ith row of A, $\bar{W}=\{A_{I,J}\,|\,J\in[0,N-1]\}$, uniformly and independently to the Ith row of V, $\{V_{I,j}\,|\,j\in[0,n-1]\}$. For each $I\in[0,M-1]$ we can use $(*)$ to bound the weight of the largest bin, $\max_j\{|V_{I,j}|\}$, by recalling that each $|A_{I,J}|$ is at most one and $|\bar{W}|$ is simply $|A_{I,*}|$, which is at most R. Hence,

$$\Pr_{\mathcal{F}_{N,n}}\left\{\max_{j\in[0,n-1]}\{|V_{I,j}|\}\geqq\frac{R}{n}(1+\varepsilon_2)\right\}\leqq n\exp\left\{-\frac{R}{n}\varepsilon_2h(\varepsilon_2)\right\}.$$

Applying this to each row I and using the union bound we get

$$\Pr_{\mathcal{F}_{N,n}}\left\{\max_{\substack{I\in[0,M-1]\\ j\in[0,n-1]}}\{|V_{I,j}|\}\geqq\frac{R}{n}(1+\varepsilon_2)\right\}\leqq Mn\exp\left\{-\frac{R}{n}\varepsilon_2h(\varepsilon_2)\right\}.$$

We complete the assignment of submatrices of A to processors by making the row assignments. Rows of V are assigned to rows of P. That is, $P_{i,*}$ is the submatrix composed of rows of V, $V_{I,*}$, such that $\rho(I)=i$ where ρ is drawn from $\mathscr{F}_{M,m}$. In other words, for a fixed column $j\in[0,n-1]$, the objects $\{V_{I,j}\mid I\in[0,M-1]\}$ are assigned uniformly and independently to the bins $\{P_{ij}\mid i\in[0,m-1]\}$. We can apply (*) so long as we have upper bounds on the weights $|V_{ij}|$ and the sum of the weights $\sum_{I=0}^{M-1}|V_{I,j}|$.

Assume for now that after the column assignment (but before the row assignment) we have $|V_{I,j}|\leqq v$ for all $I\in[0,M-1]$ and $j\in[0,n-1]$, and that $\sum_{I=0}^{M-1}|V_{I,j}|=|V_{*,j}|\leqq V$ for all $j\in[0,n-1]$. Call the former event $\mathscr{E}_v$, the latter event $\mathscr{E}_V$, and their conjunction $\mathscr{E}_{v,V}$. For each column $j\in[0,n-1]$, we can apply (*) to the random row assignment

$$\Pr_{\mathscr{F}_{M,m}}\left\{\max_{i\in[0,m-1]}\{|P_{i,j}|\}\geqq\frac{V}{m}(1+\varepsilon_1)\;\middle|\;\mathscr{E}_{v,V}\right\}\leqq m\exp\left\{-\frac{V}{vm}\varepsilon_1 h(\varepsilon_1)\right\}.$$

Applying this to every column, we obtain

$$\Pr_{\mathscr{F}_{M,m}}\left\{\max_{\substack{i\in[0,m-1]\\ j\in[0,n-1]}}\{|P_{i,j}|\}\geqq\frac{V}{m}(1+\varepsilon_1)\;\middle|\;\mathscr{E}_{v,V}\right\}\leqq mn\exp\left\{-\frac{V}{vm}\varepsilon_1 h(\varepsilon_1)\right\}.$$

To achieve a similarly small probability for $\max_{i,j}\{|P_{i,j}|\}$ being large without the conditioning event $\mathscr{E}_{v,V}$, we need only show that the probability that $\mathscr{E}_{v,V}$ is not true is very small. That is,

$$\begin{aligned}\Pr\{\mathscr{F}\}&=\Pr\{\mathscr{F}\mid\mathscr{E}_{v,V}\}\Pr\{\mathscr{E}_{v,V}\}+\Pr\{\mathscr{F}\mid\bar{\mathscr{E}}_{v,V}\}\Pr\{\bar{\mathscr{E}}_{v,V}\}\\&\leqq\Pr\{\mathscr{F}\mid\mathscr{E}_{v,V}\}+\Pr\{\bar{\mathscr{E}}_{v,V}\},\end{aligned}$$

where $\mathscr{F}$ is the event that $\max_{i,j}\{|P_{i,j}|\}\geqq(V/m)(1+\varepsilon_1)$. If we choose $V=(1+\varepsilon_3)T/n$ and $v=(1+\varepsilon_2)R/n$ then we already have an upper bound for $\Pr\{\bar{\mathscr{E}}_v\cup\bar{\mathscr{E}}_V\}$.

Putting everything together we obtain

$$\begin{aligned}\Pr\left\{\max_{i,j}\{|P_{i,j}|\}\geqq\frac{T}{mn}(1+\varepsilon_3)(1+\varepsilon_1)\right\}&\leqq mn\exp\left\{-\frac{T}{mR}\frac{1+\varepsilon_3}{1+\varepsilon_2}\varepsilon_1 h(\varepsilon_1)\right\}\\&\quad+Mn\exp\left\{-\frac{R}{n}\varepsilon_2 h(\varepsilon_2)\right\}\\&\quad+n\exp\left\{-\frac{T}{nC}\varepsilon_3 h(\varepsilon_3)\right\}.\end{aligned}$$

The choice $\varepsilon_1=\varepsilon_2=\varepsilon_3$ gives a more tractable (but not necessarily optimal) bound. Of course, we can do the whole analysis by conditioning on the results of the row assignment first. This gives the same bound as the one above with the roles of M and N, m and n, and R and C interchanged. The actual bound is the minimum of these two.

The proof for the case where the row and column assignments are made according to $\mathscr{G}_{M,m}$ and $\mathscr{G}_{N,m}$, respectively, is nearly identical, except that we need an extension of Bennett's inequality, due to Hoeffding [8], to the following case. As before, let $\bar{W}$ be a set of M objects $\bar{W}=(W_0,\ldots,W_{M-1})$ with weights $|W_I|$. Let $\hat{B}^{\bar{W}}$ be the set of objects obtained by randomly selecting r objects from $\bar{W}$ without replacement, and let $\ddot{B}^{\bar{W}}$ be the set of objects obtained by randomly selecting r objects from $\bar{W}$ with replacement. Note that

$$E(|\hat{B}^{\bar{W}}|)=E(|\ddot{B}^{\bar{W}}|)=(r/M)\sum_{K=0}^{M-1}|W_K|=r|\bar{W}|/M.$$

Also observe that

$$\mathrm{Var}\,(|\ddot{B}^W|) = (r/M) \sum_{K=0}^{M-1} (|W_K| - |\bar{W}|/M)^2,$$

but that

$$\mathrm{Var}\,(|\hat{B}^W|) = ((M-r)/(M-1))\,\mathrm{Var}\,(|\ddot{B}^W|).$$

Bennett-Hoeffding inequality. For any $d \geqq 0$ such that $|W_I| - |\bar{W}|/M \leqq d$ for all $I \in [0, M-1]$ and any $\gamma \geqq 0$

$$\Pr\{|\hat{B}^W| \geqq E(|\hat{B}^W|) + \gamma\} \leqq \exp\left\{-\frac{\gamma}{d} h\left(\frac{d\gamma}{\mathrm{Var}\,(|\ddot{B}^W|)}\right)\right\}.$$

We apply this bound to the situation where we place exactly $\lceil M/m \rceil$ objects into bins $\hat{B}_i^W$, $i \in [0, (M \bmod m) - 1]$, and exactly $\lfloor M/m \rfloor$ objects into bins $\hat{B}_i^W$, $i \in [M \bmod m, m-1]$. Formally, $\hat{B}_i^W = \{W_I \mid I \in \rho^{-1}(i)\}$ where ρ has distribution $\mathscr{G}_{M,m}$. By symmetry we know that each $\hat{B}_i^W$, $i \in [0, (M \bmod m) - 1]$, has the same distribution as $\hat{B}^W$ above with $r = \lceil M/m \rceil$ and each $\hat{B}^W$, $i \in [M \bmod m, m-1]$, has the same distribution as $\hat{B}^W$ above with $r = \lfloor M/m \rfloor$. Using algebra similar to that used for deriving (*) from Bennett's inequality, it is straightforward to derive the following bound from the Bennett-Hoeffding inequality.

For any $W \geqq |\bar{W}|$, any $w > 0$ such that for all $I \in [0, M-1]$, $W_I \leqq w$, and $\varepsilon \geqq 0$,

$$\Pr_{\mathscr{G}_{M,m}}\left\{\max_{i \in [0, m-1]} \{|\hat{B}_i^W|\} \geqq W \frac{\lceil M/m \rceil}{M}(1+\varepsilon)\right\} \leqq m \exp\left\{-\frac{W}{w}\frac{\lceil M/m \rceil}{M} \varepsilon h(\varepsilon)\right\}.$$

This is the same bound as (*) when we substitute the m' and n' as previously defined. Hence, the remainder of the proof follows exactly as before. □

3. Data-parallel sparse matrix–vector multiplication. So far, we have analyzed randomized algorithms for the balanced assignment of nonzeros of a sparse matrix to a rectangular processor array. Such assignments preserve the alignment of matrix rows and columns for the design of efficient parallel sparse matrix routines. In this section we consider the design of the basic sparse matrix-vector multiplication kernels for parallel processor arrays under the restrictive conditions of the data-parallel single instruction multiple data (SIMD) machine model. Program-parallel multiple instruction multiple data (MIMD) machines are more powerful, and naturally include the SIMD model.

The minimum characteristics of a data-parallel computer model required here are:

1. There are p PEs interconnected by a communication network. Every PE has its own, identically organized local memory. It is assumed that the network is configured as a (virtual) two-dimensional rectangular grid, with efficient communication among the PEs along any row or column of the grid.

2. Each PE can be in the active or inactive state independently, depending on the local data, and this may change from instruction to instruction.

3. There is a separate processor (controller) executing the program and broadcasting instructions which are synchronously evaluated in all active PEs. It is assumed that the *indirect addressing* feature is available: The PEs can store local pointers to their local memories, thus each PE can access a different memory location in a single instruction. We will concentrate on the second (i.e., random permutation) assignment scheme. While both schemes produce balanced load under the same assumptions, and,

with balanced load, lead to similar matrix algorithms, only the second scheme guarantees the upper limit on the dimension of the submatrices allocated to each processor. This simplifies memory management and algorithm design for SIMD computers.

Therefore, suppose that for a large sparse matrix A, the random row and column permutations result in an acceptably balanced distribution of the nonzeros of A to the PEs. If the row permutation is represented by an $M \times M$ matrix P, and the column permutation is represented by an $N \times N$ matrix Q, then the assignment of nonzeros to PEs considered in § 2 can be written as $(PAQ^T)_{I,J} \rightarrow PE(I \bmod m, J \bmod n)$. In most matrix problems one may do the computations with PAQ^T instead of A and undo the permutations at the end. This is straightforward; therefore, from now on, we will ignore the permutations for the sake of simplicity, and we will set $P = \mathbf{I}_M$ and $Q = \mathbf{I}_N$.

Each PE stores the nonzeros of a sparse submatrix of A in a compressed data structure. The choice of compression scheme may be dictated by the PE architecture, available local memory, or other considerations [13], [14]. Here we use a simple symmetric scheme which allows for fast computation of both right and left multiplications, $y = Ax$ and $x^T = y^T A$. The processors' memory is therefore organized as follows: The nonzeros allocated to $PE(i,j)$ are stored in three aligned arrays, $a[k]$, $r[k]$, $c[k]$, $k = 0, 1, \ldots, |P_{ij}| - 1$, where a is the matrix element and r and c are its row and column indices, respectively.

It is not efficient to store dense vectors as matrices with one row or column. For communication efficiency and good load balance, it is preferable to distribute the components of a vector $x = (x_0, x_1, \ldots, x_{L-1})$ to processors according to a multilayer lexicographic scheme. In each PE define a local array $u[\,]$ of length $\lceil L/mn \rceil$. For row-wise lexicographic storage renumber the processors with a single index, so that $PE(i,j)$ gets the index $k = i \cdot n + j$, and place the component x_J in array element $u[\lfloor J/mn \rfloor]$ in the PE with index $k = J \bmod mn$, for $J = 0, 1, \ldots, L-1$. For column-wise lexicographic placement renumber the processors so that $PE(i,j)$ gets the index $k = j \cdot m + i$, and proceed as before.

We shall discuss only the routine for computation of $y = Ax$ in detail. Obvious modifications are required for the computation of $y^T A$, and for the extension to blocks of dense vectors. The vector x is distributed among processors in a row-wise lexicographic order and stored in the local arrays $u[\,]$, while vector y is distributed in the column-wise lexicographic order, and stored in the local arrays $v[\,]$. The multiplication routine requires an auxiliary local accumulator array acc[ ] of length $|P_{ij}|$, and an auxiliary local buffer array buf[ ] in each $PE(i,j)$, and proceeds in several phases.

For transparency, the pseudocode below is written for the case when the PE array dimensions divide the matrix dimensions, i.e., $M = rm$ and $N = sn$, and each PE has sufficient memory for the buffer array of length $\max(r, s)$. We note that indirect addressing is critical for data-parallel execution of the scatter and gather steps.

Matrix–vector multiplication $y = Ax$

```
1. Distribute vector components
      for $k = 0, \ldots, s-1$
          in parallel in each array column $j = 0, \ldots, n-1$
              $temp = k \bmod m$,
              $PE(temp, j)$ sends $u[\lfloor k/m \rfloor]$ to all PEs in column $J$,
              every PE copies received value in buf$[k]$.
2. Scatter: every $PE(i,j)$ in parallel
      for $k = 0, \ldots, |P_{ij}| - 1$
          acc$[k] \leftarrow$ buf$[(c[k] - j)/n]$.
```

```
3. Multiply: every $PE(i,j)$ in parallel
       for $k=0,\ldots,|P_{ij}|-1$
           $\mathrm{acc}[k] \leftarrow \mathrm{acc}[k] * a[k]$.
4. Gather: every $PE(i,j)$ in parallel
       for $k=0,\ldots,r-1$
           $\mathrm{buf}[k]=0$,
       for $k=0,\ldots,|P_{ij}|-1$
           $temp=(r[k]-i)/m$,
           $\mathrm{buf}[temp] \leftarrow \mathrm{buf}[temp]+\mathrm{acc}[k]$.
5. Row sums
       for $k=0,\ldots,r-1$
           in parallel in each array row $i=0,\ldots,m-1$
               compute the sum of $\mathrm{buf}[k]$ along row $i$,
               copy the sum to $v[\lfloor k/n \rfloor]$ in $PE(i, k \bmod n)$.
```

When the routine completes execution, the local array element $v[l]$ in $PE(i,j)$ stores the vector component $y_{lmn+jm+i}$, as determined by column-wise lexicographic order.

The number of parallel operations is as follows: $\lceil N/m \rceil$ vector copy steps; $\max_{i,j} |P_{ij}|$ each of the scatter, multiply, and gather steps; $\lceil M/m \rceil$ row sum evaluations. In practice, it may be more efficient to employ systolic techniques for the first and last stages, rather than use broadcast along array columns and segmented scan-adds, respectively. The distribute/scatter and gather/sum steps may be iterated if there is not enough memory for a long buffer array. Multiple iterations can be efficiently managed with pointers when the arrays a, r, and c are sorted in the order of increasing row index, $r[0] \leqq r[1] \leqq \cdots \leqq r[|P_{ij}|-1]$, and in the order of increasing column index via an auxiliary local pointer array $p[i]$ such that $c[p[0]] \leqq c[p[1]] \leqq \cdots \leqq c[p[|P_{ij}|-1]]$.

3.1. A practical implementation. In order to demonstrate the practicality of data structures and algorithms proposed in this report, we have implemented the matrix multiplication routines and load balancing on a commercially available computer, the MasPar MP-1216. This is a data-parallel machine with 16,384 RISC processors. Each processor has 64 kbytes of local memory, and operates on four-bit-wide data fields, with floating point instructions implemented in microcode. There are two separate communication networks: a two-dimensional toroidal mesh, and a global router. Only the mesh network is used in matrix calculations, and on the MP-1216 the PEs are connected as a 128×128 array. The programs have been written in MPL [11], which is a data-parallel extension of the ANSI standard C language [10]. Several parallel algorithms for dense matrix multiplication have been implemented and analyzed on this computer in [2].

We have found that the performance of the load balancing algorithm is much better in practice than guaranteed by our theorem. The reason for this is that in the proof some dependencies have been bounded by overcounting, and some bounds have been relaxed to obtain a compact final formula. An extended empirical study of our assignment scheme for a variety of structures and sizes is beyond the scope of this report, nonetheless, as an illustration we do present data for some large matrices, both unstructured and highly structured. Let $P_0=\lceil T/mn \rceil$ be the perfectly balanced load per PE. For the matrices described below we have estimated the distribution function $\Pr\{\max_{i,j} |P_{ij}|/P_0<\lambda\}$, which succinctly illustrates the performance of the randomized allocation algorithm. The probability distribution is the $\mathcal{G}_{M,m} \times \mathcal{G}_{N,n}$ of § 2, corresponding to independent randomly drawn row and column permutations. Empirical distribution functions were obtained from a few hundred independent assignments for each matrix.

For the first example we take a 25,629 × 56,530 matrix representing the frequencies of words occurring in more than two articles from the Academic American Encyclopedia. This is a sparse, unstructured matrix with $T = 2{,}843{,}956$ nonzeros, $C = 13{,}904$, and $R = 2{,}168$. However, despite the presence of some quite dense rows and columns the random permutation assignment works reasonably well. An estimate of $\Pr\{\max_{ij} |P_{ij}|/P_0 < \lambda\}$ for the 128 × 128 MasPar array is shown in Fig. 1. We see that a maximum load of at most $2P_0$ can be achieved in one attempt with probability about 0.22. This probability increases sharply when the requirement on the maximum load is relaxed. Qualitatively similar results have been obtained for other large sparse matrices representing word frequencies in different document databases.

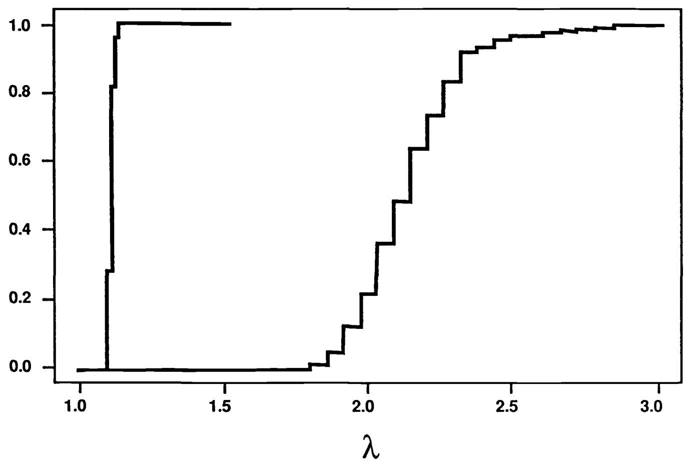


FIG. 1. *Estimated probability distributions Pr* $\{\max_{ij} |P_{ij}|/P_0 < \lambda\}$ *for the band matrix (left) and the word frequency matrix (right) described in the text.*

For a completely different example we consider a square 200,000 × 200,000 banded matrix with a half-bandwidth of 100. Although for such special matrices we would rather design a different distributed data structure, it is instructive to see the power of randomization: A direct tiling mapping of this matrix onto the 128 × 128 PE array would produce $\max_{i,j} |P_{ij}| = 151{,}350$ compared to $P_0 = 1221$. However, the random permutation assignment does very well (see Fig. 1): with probability of success over 0.99, we obtain an assignment that deviates from the perfectly balanced load by less than 15 percent.

In order to assess the performance of our implementation of the sparse matrix-vector multiplication routine, and to estimate the fraction of time spent on interprocessor communication and nonnumerical operations, we compare the performance of the routine to the machine's peak floating point computing speed. All performance figures are for the double precision (64 bits) floating point format for matrix and vector components. According to custom, we also characterize the routine's performance in terms of the number of floating point operations per second (flops), including all nonnumerical operations in the time measurement.

For a standard of the machine peak rate we consider two array operations $c[i] \leftarrow a[i]+b[i]$ and $c[i] \leftarrow a[i]*b[i]$ executing in parallel in the PEs without interprocessor communication. The timing of these instructions for long arrays on a 16,384-processor MasPar gives the average peak rate of 250 Mflops (register operations are about twice as fast, but not proper for comparisons with routines involving array references). For the sparse matrix-vector multiplication routine, the flop rate is determined by the number of nonzeros T, and in accordance with [7], is defined as $2T/\tau$, where τ is the routine execution time. The highest performance has been achieved with the perfectly balanced PE load (i.e., $\max_{ij} |P_{ij}| = T/16{,}384$) for large dense matrices cast in the sparse data structure, giving 116 Mflops, that is, about 45 percent of the peak machine rate determined above. For sparse matrices the ratio $\max_{ij} |P_{ij}|/P_0$ quantifies the performance loss resulting from load imbalance.

Our fastest implementation of the sparse matrix-vector multiplication on the MasPar utilizes only the nearest-neighbor communications in the vector distribution and row sum collection stages. We have analyzed its performance, obtaining a formula for the execution time τ,

$$\tau = \tau_1 \cdot m \cdot \lceil N/mn \rceil + \tau_2 \cdot \max_{i,j} |P_{ij}| + \tau_3 \cdot n \cdot \lceil M/mn \rceil.$$

The first term accounts for the vector distribution; the second for the scatter, multiply, and gather steps; and the last for the row sum collection. The expected operation times τ_i, measured on a 128×128 machine, are $\tau_1 = 17$ μs, $\tau_2 = 290$ μs, and $\tau_3 = 84$ μs (microseconds). The formula for τ predicts quite accurately the actual execution time for arbitrary matrices and imperfectly balanced loads, with a small uncertainty due to the dependence of the course of the scatter and gather operations on the data. For instance, for the first word frequency matrix described above, we obtained 48 Mflops, and for the banded matrix we obtained 70 Mflops. On this machine for a fixed T, the flop rate decreases with increasing matrix dimensions M and N; thus, for extremely sparse matrices with very large dimensions another algorithm could offer better performance.

Acknowledgments. We thank Tom Leighton and Michael Berry for useful discussions and Sue Dumais for examples of word-frequency matrices.

REFERENCES

[1] G. Bennett, *Probability inequalities for the sum of independent random variables*, J. Amer. Statist. Assoc., 57 (1962), pp. 33-45.

[2] P. Bjørstad, F. Manne, T. Sørevik, and M. Vajteršic, *Efficient matrix multiplication on* SIMD *computers*, University of Bergen Tech. Rep., Bergen, Norway, 1991.

[3] E. Dekel, D. Nassimi, and S. Sahni, *Parallel matrix and graph algorithms*, SIAM J. Comput., 10 (1981), pp. 657-675.

[4] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh, *Parallel algorithms for dense linear algebra computations*, SIAM Rev. 32 (1990), pp. 54-135.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, New York, 1979.

[6] W. M. Gentleman, *Some complexity results for matrix computations on parallel processors*, J. Assoc. Comput. Mach., 25 (1978), pp. 112-115.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.

[8] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13-30.

[9] H. J. Jagadish and T. Kailath, *A family of new efficient arrays for matrix multiplication*, IEEE Trans. Comput., 38 (1989), pp. 149-155.

[10] B. W. Kernighan and D. M. Ritchie, *The* C *Programming Language*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1988.

[11] *MasPar* MP-1 *Standard Programming Manuals*, MasPar Computer Corporation, Sunnyvale, CA, 1991.

[12] J. M. Ortega, R. G. Voigt, and C. H. Romine, *A bibliography on parallel and vector numerical algorithms*, in Parallel Algorithms for Matrix Computations, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 125–197.

[13] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press, London, 1984.

[14] Y. Saad, *Krylov subspace methods on supercomputers*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1220–1232.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 531-541, May 1993

SOME ASPECTS OF CIRCULANT PRECONDITIONERS*

THOMAS HUCKLE†

Abstract. This paper studies circulant approximations of Hermitian Toeplitz matrices that are solutions of certain minimization problems relative to the Frobenius norm, the lub_1, or the lub_∞ norms. The examinations supplement the family of the so-called optimal and superoptimal preconditioners that have been proposed for the preconditioned conjugate gradient method for solving Toeplitz systems $T_n x = b$. For the new Frobenius norm approximations it is shown that they can be computed in $O(n \log(n))$ arithmetic operations, and that the eigenvalues of the preconditioned linear equations are asymptotically clustered around 1 if T_n is a leading principal submatrix of an infinite Toeplitz matrix connected with a positive function of the Wiener class.

Key words. Toeplitz matrix, circulant matrix, preconditioned conjugate gradient method

AMS(MOS) subject classifications. 65F10, 65F15

0. Introduction. In this paper we derive some new properties of circulant preconditioners for the solution of linear Toeplitz equations. A matrix T_n is a Hermitian Toeplitz matrix if

$$T_n = \begin{pmatrix} t_0 & t_1 & \cdots & \cdots & t_{n-1} \\ \bar{t}_1 & t_0 & t_1 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \bar{t}_1 & t_0 & t_1 \\ \bar{t}_{n-1} & \cdots & \cdots & \bar{t}_1 & t_0 \end{pmatrix} = T_n(t_0, t_1, \ldots, t_{n-1}), \tag{1}$$

and a matrix C is a Hermitian circulant matrix if $C = T_n(c_0, c_1, c_2, \ldots, \bar{c}_2, \bar{c}_1)$. Strang in [14] proposed using the preconditioned conjugate gradient (PCG) method for solving symmetric positive definite Toeplitz systems. By the fast Fourier transform, linear equations with circulant matrices can be solved in $O(n \log(n))$ operations, and every circulant matrix has a diagonalization of the form $C = F_n^H \Lambda F_n$ with the Fourier matrix (see [7])

$$F_n^H := \frac{1}{\sqrt{n}} (\exp(2\pi ijk/n))_{j,k=0}^{n-1} = \frac{1}{\sqrt{n}} (w^{jk})_{j,k=0}^{n-1}, \tag{2}$$

where $w = \exp(2\pi i/n)$.

An infinite Toeplitz matrix $T = (t_{i-j})_{i,j=1}^{\infty}$ is connected with a function

$$f(\theta) := \sum_{j=-\infty}^{\infty} t_j e^{-ij\theta}, \qquad \theta \in [0, 2\pi],$$

which is said to be in the Wiener class if f is an l_1 function, thus $\sum_{j=-\infty}^{\infty} |t_j| \leqq M < \infty$. If the Toeplitz matrices T_n are leading principal submatrices of an infinite Toeplitz matrix T with underlying function f positive and in the Wiener class, the eigenvalues of the preconditioned systems $C^{-1}T_n$ are asymptotically clustered around 1 for many different circulant approximations C on T_n (see [1]-[4], [6], [10]-[13], [15], and [16]). Hence, the number of iterations in the PCG method is bounded independent of n in this case.

* Received by the editors August 7, 1991; accepted for publication (in revised form) May 11, 1992.

† Institut für Angewandte Mathematik und Statistik, Universität Würzburg, D-8700 Würzburg, Germany.

1. Frobenius norm approximations. In [6], T. Chan introduced the optimal circulant Frobenius norm approximation C_F for a given Hermitian matrix A. Every element c_k of C_F is the mean value of the corresponding elements on a kth diagonal of A. For the following, using the notation of relation (1), let us denote

$$\text{(3)} \qquad \operatorname{Circ}(A) := T_n(c_0, c_1, c_2, \dots, \bar{c}_2, \bar{c}_1) \quad \text{with } c_k := \frac{1}{n} \sum_{j=1}^{n} a_{j,j+k \bmod n}.$$

Then $C_F = \operatorname{Circ}(A)$. The convergence of the PCG method depends on the spectrum of $C^{-1/2}AC^{-1/2}$ or $C^{-1}A$. Hence, it seems to be more appropriate to look for a positive definite circulant matrix that minimizes $\|I - C^{-1}A\|$ for a certain norm. This leads to the superoptimal Frobenius norm approximation of Tyrtyshnikov [3], [15], [16] $C_T = \operatorname{Circ}(A^2) * \operatorname{Circ}(A)^{-1}$, which minimizes $I - C^{-1}A$ in the Frobenius norm. Here, $*$ denotes the standard matrix or scalar multiplication.

But this is not the true optimal approximation, because the Hermitian matrix which is used in the PCG method is $C^{-1/2}AC^{-1/2}$, and thus one should better minimize $\|I - C^{-1/2}AC^{-1/2}\|$ in some norm. For the Frobenius norm we can again solve this problem.

THEOREM 1. *Let A be a Hermitian positive definite $n \times n$ matrix. Let $B = F_nAF_n^H$, $D = B \circ \bar{B}$ be the Hadamard product of B and $\bar{B}$, $b = \operatorname{diag}(B)$, and $x = D^{-1}b$. Then D is positive definite. If all components of the vector x are positive, define*

$$\Lambda = \operatorname{diag}\left(\frac{1}{x_1}, \dots, \frac{1}{x_n}\right), \qquad C_B := F_n^H \Lambda F_n,$$

and it holds that

$$\|I - C_B^{-1/2}AC_B^{-1/2}\|_F = \min_{C \text{ circulant}} \|I - C^{-1/2}AC^{-1/2}\|_F.$$

Proof. With $C = F_n^H \Lambda F_n$,

$$\|I - C^{-1/2}AC^{-1/2}\|_F^2 = \|I - \Lambda^{-1/2}B\Lambda^{-1/2}\|_F^2 = 1 - 2\frac{b_{11}}{\lambda_1} + \sum_{j=1}^{n} \frac{|b_{1j}|^2}{\lambda_1 \lambda_j} + \cdots$$
$$= f(\lambda_1, \dots, \lambda_n).$$

The condition $\nabla f = 0$ is equivalent to the equations

$$\text{(4)} \qquad b_{kk} = \sum_{j=1}^{n} \frac{|b_{k,j}|^2}{\lambda_j}, \qquad k = 1, \dots, n.$$

Therefore, we get the minimal Frobenius norm approximation by solving this system of linear equations in $x = (1/\lambda_1, \dots, 1/\lambda_n)$ if all components of x are positive. Note that with A, the matrix D is also positive definite [9, Thm 5.3.1], and that b_{ii} are the eigenvalues of C_F with $\lambda_{\min}(A) \leqq b_{ii} \leqq \lambda_{\max}(A)$ [11], [12]. □

To show that this new preconditioner is competitive with C_F for Toeplitz matrices connected with positive functions of the Wiener class, we prove in the following that under this assumption

(a) the matrices B and D can be computed in $O(n \log(n))$,

(b) equation (4) can be solved in $O(n \log(n))$,

(c) the solution of (4) is positive for n large enough,

(d) C_F and C_B are asymptotically equivalent, and thus the eigenvalues of $C_B^{-1}T_n$ are clustered around 1 for large n [1], [2], [3].

First, let us consider (c) and (d).

THEOREM 2. *If $A = T_n$ is a Hermitian Toeplitz matrix with an underlying positive function f of the Wiener class, then*

$$\|C_B - C_F\|_2 \to 0 \quad \text{for } n \to \infty,$$

and thus C_B and C_F are asymptotically equivalent relative to the spectral norm. Hence, for n large enough, C_B and C_B^{-1} are well defined and of bounded spectral norm.

Proof. First, note that the diagonal elements of B are the eigenvalues of C_F and are contained in the interval $[m, M]$ with $0 < m = \min f \leqq M = \max f \leq \infty$ [1], [11], [12]. For the elements of B,

$$\begin{aligned} b_{i,j} &= \frac{1}{n} \sum_{k,m=1}^{n} a_{k,m} \bar{w}^{(i-1)(k-1)} w^{(j-1)(m-1)} \\ &= \frac{1}{n} \sum_{r=1-n}^{n-1} \bar{t}_r \bar{w}^{j-i+(i-1)r} \sum_{m=\max\{1,1-r\}}^{\min\{n,n-r\}} w^{(j-i)m}. \end{aligned} \tag{5}$$

Hence, we get

$$|b_{i,i}|^2 = \frac{1}{n^2} \sum_{r,k=1-n}^{n-1} t_k \bar{t}_r (n-|r|)(n-|k|) w^{(i-1)(k-r)}$$

and

$$\begin{aligned} \sum_{j=1}^{n} |b_{i,j}|^2 &= \frac{1}{n^2} \sum_{r,k=1-n}^{n-1} t_k \bar{t}_r w^{(i-1)(k-r)} \sum_{m_1=\max\{1,1-r\}}^{\min\{n,n-r\}} \sum_{m_2=\max\{1,1-k\}}^{\min\{n,n-k\}} w^{i(m_2-m_1)} \sum_{j=1}^{n} w^{(m_1-m_2)j} \\ &= \frac{1}{n^2} \sum_{r,k=1-n}^{n-1} t_k \bar{t}_r w^{(i-1)(k-r)} \sum_{m=\max\{1,1-r,1-k\}}^{\min\{n,n-r,n-k\}} n. \end{aligned}$$

Therefore, with the notation $B = (b_1, b_2, \ldots, b_n)$,

$$\begin{aligned} \|b_i\|_2^2 - b_{i,i}^2 &= \sum_{j \neq i, j=1}^{n} |b_{i,j}|^2 \\ &= \sum_{r,k=1-n}^{n-1} t_k \bar{t}_r w^{(i-1)(k-r)} \\ &\quad \cdot \left(\frac{[\min\{n, n-r, n-k\} - \max\{0, -r, -k\}]_+}{n} - \frac{(n-|r|)(n-|k|)}{n^2} \right) \\ &= \frac{1}{n^2} \sum_{r,k=1-n}^{n-1} t_k \bar{t}_r w^{(i-1)(k-r)} c_{r,k} \end{aligned}$$

with

$$c_{r,k} = \begin{cases} \min\{(n-|r|)|k|, (n-|k|)|r|\} & \text{for } rk \geqq 0, \\ -|rk| & \text{for } rk < 0 \text{ and } |k|+|r| \leqq n, \\ -(n-|r|)(n-|k|) & \text{for } rk < 0 \text{ and } |k|+|r| \geqq n, \end{cases}$$

and $[x]_+ := \max\{0, x\}$. Thus $|c_{r,k}| \leqq Mn$ for $|r| \leqq M$ or $|k| \leqq M$, M independent of n, and

$$\phi(n) := \max\{\|b_i\|_2^2 - b_{i,i}^2\} \to 0 \quad \text{for } n \to \infty. \tag{6}$$

For a proof of (6), see also the Corollary to Theorem 4 in [3]. With the notations of Theorem 1, $\|D-\operatorname{diag}(b_{11}^2,\ldots,b_{nn}^2)\|_\infty$ tends to 0 for increasing n, and therefore $\|D\|_\infty$, $\|D^{-1}\|_\infty$, and $\|x\|_\infty=\|D^{-1}b\|_\infty$ are bounded independent of n. Hence,

$$\begin{aligned}\max_i\left\{\left|b_{ii}-\frac{b_{ii}^2}{\lambda_i}\right|\right\} &= \|b-\operatorname{diag}(b_{11}^2,\ldots,b_{nn}^2)*x\|_\infty\\ &= \|(D-\operatorname{diag}(b_{11}^2,\ldots,b_{nn}^2))*x\|_\infty\\ &\leqq \|D-\operatorname{diag}(b_{11}^2,\ldots,b_{nn}^2)\|_\infty\|x\|_\infty\end{aligned}$$

tends to 0 for increasing n, and

$$0<m_1\leqq\lambda_i\leqq M_1<\infty,\qquad i=1,\ldots,n,$$

for large n. □

THEOREM 3. *For a Hermitian Toeplitz matrix T_n, the matrices $B=F_nT_nF_n^H$ and $D=B\circ\bar B$ can be computed in $O(n\log(n))$. Furthermore, for a vector a, the evaluation of $B*a$ and $D*a$ take $O(n\log(n))$ operations.*

Proof. Using (5), for $i\neq j$,

$$\begin{aligned}nb_{i,j}&=\sum_{r=1}^{n-1}\bar t_r\bar w^{j-i+(i-1)r}\sum_{m=1}^{n-r}w^{(j-i)m}+\sum_{r=1-n}^{-1}\bar t_r\bar w^{j-i+(i-1)r}\sum_{m=1-r}^{n}w^{(j-i)m}\\ &=\frac{1}{1-w^{j-i}}*(g_i-g_j+\bar g_j-\bar g_i),\end{aligned}$$

with $g_i:=\sum_{r=1}^{n-1}\bar t_r(\bar w^{(i-1)})^r$, $i=1,\ldots,n$, and $g=(g_1,\ldots,g_n)^T$ can be computed by the fast Fourier transform applied to $(0,\bar t_1,\ldots,\bar t_{n-1})$ in $O(n\log(n))$. Define the circulant Hermitian matrix

$$T_w:=\frac{1}{n}*T_n\left(0,\frac{1}{1-w},\ldots,\frac{1}{1-w^{n-1}}\right),$$

the purely imaginary matrix $G:=(g-\bar g)*(1,\ldots,1)$, and the rank-2 matrix $E:=G+G^H$. Then,

$$B=(E\circ T_w)+\operatorname{diag}(b_{11},\ldots,b_{nn})$$

and

$$D=(E\circ\bar E)\circ(T_w\circ\bar T_w)+\operatorname{diag}(b_{11}^2,\ldots,b_{nn}^2)$$

with $\operatorname{rank}(E\circ\bar E)\leqq 4$ (see [9, Thm 5.1.7]). Furthermore, with $e:=(1,\ldots,1)^T$, $h:=g-\bar g$, and $D(a):=\operatorname{diag}(a_1,\ldots,a_n)$ for a given vector a, we get

$$B*a=((h*e^T+e*h^H)\circ T_w)*a+\operatorname{diag}(b_{11},\ldots,b_{nn})*a,$$

and with [9, Lemma 5.1.3],

$$(T_w\circ(h*e^T))*a=\operatorname{diag}(T_w*D(a)*(e*h^T))=\operatorname{diag}((T_w*a)*h^T),$$

and

$$(T_w\circ(e*h^H))*a=\operatorname{diag}(T_w*D(a)*(\bar h*e^T))=T_w*(a\circ\bar h)$$

can be computed in $O(n\log(n))$. Obviously, the same holds for $D*a$. □

Remark. (1) In view of Theorem 4, it seems to be more efficient to apply the PCG method not to the original linear equations $T_nx=b$, but to the corresponding system $B(F_nx)=F_nb$. Each iteration of the PCG method with preconditioner C and matrix

T_n takes six Fourier transforms, but with diag $(b_{11}, \ldots, b_{nn})$ and B, it takes only four Fourier transforms for evaluating $T_w * a$ and $T_w * (a \circ \bar{h})$ with circulant T_w. Note that in this formulation the preconditioner is the diagonal of the underlying matrix of the linear system.

(2) A similar representation of a Toeplitz matrix by means of the Hadamard product of a rank-2 matrix with a unitary circulant matrix C_0 is given in [15]. The first row of C_0 is

$$\frac{2}{n(1-\sigma w^k)}, \quad k=0, \ldots, n-1, \quad \text{with } \sigma = \exp(\pi i/n),$$

and C_0 has the eigenvalues $\mu_k = \sigma^k$, $k=0, \ldots, n-1$. The eigenvalues of T_w are $\lambda_k = -\frac{1}{2} + (2k+1)/2n$, $k=0, \ldots, n-1$. Hence, μ_k and λ_k are connected by $\mu_k = \exp((\lambda_k - \lambda_0)\pi i)$, $k=0, \ldots, n-1$.

For general Hermitian positive definite matrix A, the matrices B and D are general Hermitian positive definite matrices. Thus it takes $O(n^2 \log(n))$ operations to compute B by $2n$ fast Fourier transforms and, in addition, $O(n^2)$ operations for D. But there are interesting cases for which the matrix D is of special form. Consider $A = C + E$, with C circulant and E of low rank. Then $B = \Lambda + F_n E F_n^H = \Lambda + \tilde{E}$ with rank $(\tilde{E}) =$ rank (E), and

$$D = \Lambda \circ \Lambda + \Lambda \circ \tilde{E} + \tilde{E} \circ \Lambda + \tilde{E} \circ \tilde{E} = \text{diagonal} + \tilde{E} \circ \tilde{E} = \text{diagonal} + \hat{E}$$

and rank $(\hat{E}) \leqq$ rank $(\tilde{E})^2$ (see [9, Thm. 5.1.7]). Then D can be computed in $O(n \log(n))$ operations, and $Dx = b$ can be solved in $O(n)$. For example, let

$$A = T_n(2, -1, 0, \ldots, 0) = T_n(2, -1, 0, \ldots, 0, -1) + e_1 e_n^T + e_n e_1^T$$

(see [5]). In this case, $D = \text{diagonal} + \hat{E}$ with rank $(\hat{E}) \leqq 4$ can be computed in $O(n)$ in view of the special structure of A. Unfortunately, numerical computations show that for this matrix A the vector x in Theorem 1 has exactly one negative component for many n. This is caused by the fact that the underlying l_1 function f is only nonnegative.

If x has negative components, we can define positive definite preconditioners by setting

$$\lambda_i := |1/x_i|, \quad C_{Bs} := F_n^H \operatorname{diag}(\lambda_i) F_n \quad \text{if } x_i \neq 0,$$

or better,

$$\lambda_i := \begin{cases} 1/x_i & \text{if } x_i > 0, \\ b_{ii} & \text{else,} \end{cases} \quad C_{Bb} := F_n^H \operatorname{diag}(\lambda_i) F_n.$$

In the last section we will give numerical results for C_{Bs} and C_{Bb}.

THEOREM 4. *For a Hermitian Toeplitz matrix with an underlying positive function of the Wiener class,* (4) *can be solved in* $O(n \log(n))$ *operations.*

Proof. Following (6), the spectrum of the matrix $R := \operatorname{diag}(b_{11}^{-2}, \ldots, b_{nn}^{-2}) * D$ is contained in the interval $[1 - \phi(n), 1 + \phi(n)]$ with $\phi(n) \to 0$ for $n \to \infty$. Let us use the PCG method for solving $Dx = b$ with preconditioner diag $(b_{11}^2, \ldots, b_{nn}^2)$. Note that for a vector a, $D * a$ can be computed in $O(n \log(n))$ operations in view of Theorem 3. The condition number κ of R is bounded by

$$\kappa \leqq \frac{1 + \phi(n)}{1 - \phi(n)},$$

and thus for

$$e_k := \sqrt{(A^{-1}b - x_k)^H A(A^{-1}b - x_k)},$$

the error after the kth step (see [8]),

$$\frac{e_k}{e_0} \leqq \left(\frac{1-\sqrt{(1+\phi(n))/(1-\phi(n))}}{1+\sqrt{(1+\phi(n))/(1-\phi(n))}}\right)^{2k} = \left(\frac{1-\sqrt{1-\phi(n)^2}}{\phi(n)}\right)^{2k} = \left(\frac{\phi(n)}{2} + O(\phi(n)^3)\right)^{2k}.$$

Hence, $e_k/e_0 \leqq \varepsilon$ is fulfilled if asymptotically

$$\left(\frac{\phi(n)}{2}\right)^{2k} \leqq \varepsilon,$$

or $\phi(n) \leqq 2 * \varepsilon^{1/(2k)}$. Therefore, the number of iterations of the PCG method for computing the vector x is bounded by $O(1/|\log(\phi(n))|)$, and the total number of operations for solving (4) is $O(n \log(n))$. □

We can consider a similar minimization problem that leads to another well-defined circulant approximation to a given matrix A. Let us look for the solution of

$$(7) \qquad \min_{C \text{ circulant}} \|C^{1/2} - A * C^{-1/2}\|_F.$$

Because of

$$\|C^{1/2} - A * C^{-1/2}\|_F = \|\Lambda^{1/2} - B * \Lambda^{-1/2}\|_F$$
$$= \lambda_1 - 2b_{11} + \frac{\|b_1\|_2^2}{\lambda_1} + \cdots,$$

the solution of (7) is given by the circulant matrix C_{F2} with eigenvalues $\lambda_i = \|b_i\|_2$, and we get with (3),

$$C_{F2} = \text{Circ}\,(A^2)^{1/2},$$

which is always defined, positive semidefinite, and asymptotically equal to C_F if $A = T_n$ is related to a positive function of the Wiener class. In view of the connection between C_T, C_F, and C_{F2}, C_{F2} can be computed in $O(n \log(n))$ [3], [15], [16], and the clustering property that holds for Toeplitz matrices with underlying positive l_1 function is true for $C_{F2}^{-1}T_n$ as well [1], [2], [3].

Note that the various Frobenius norm approximations fulfill

$$\text{Circ}\,(A) = \text{Circ}\,(C_F) = \text{Circ}\,(C_T * A^2) = \text{Circ}\,(A * C_B^{-1} * A),$$
$$\text{Circ}\,(C_{F2}^2) = \text{Circ}\,(A^2).$$

The equation for C_B follows from $\text{diag}\,(B * \text{diag}\,(1/\lambda_1, \ldots, 1/\lambda_n) * \bar{B}) = \text{diag}\,(B)$ [9, Lemma 5.1.3] and Theorem 2 in [3].

2. Approximations in other norms. Looking for a circulant matrix that minimizes $\|C - A\|_2$ is equivalent to minimizing

$$\|\Lambda - B\|_2$$

with a diagonal matrix Λ and $B = F_n A F_n^H$. In this section we will determine some circulant approximations that correspond to an (in some sense) optimal diagonal matrix Λ. First, let us remark that the Frobenius norm approximation C_F not only solves $\min \|\Lambda - B\|_F = \|C - A\|_F$, but also $\min \|\Lambda - B\|_1$ and $\min \|\Lambda - B\|_\infty$. A similar property is possessed by the circulant approximation of Strang [14], which for Toeplitz matrices solves $\min \|C - T_n\|_1 = \min \|C - T_n\|_\infty$.

We can also determine superoptimal diagonal approximations in the $\|\cdot\|_1$ norm.

THEOREM 5. *Let A be a Hermitian positive definite matrix. If $\|b_i\|_1 \leqq 2 * b_{ii}$, $i = 1, \dots, n$, then the matrix $\Lambda_F := F_n C_F F_n^H$ is a solution of*

$$\min_{\Lambda \text{ diagonal}} \|I - \Lambda^{-1} B\|_1.$$

Proof. It holds that

$$\|I - \Lambda^{-1}B\|_1 = \max_{i=1}^{n} \left\{ \left|1 - \frac{b_{ii}}{\lambda_i}\right| + \frac{\|b_i\|_1 - b_{ii}}{\lambda_i} \right\}.$$

For $b_{ii} \geqq \lambda_i$ the ith term takes its minimal value if $\lambda_i = b_{ii}$. For $b_{ii} < \lambda_i$ and $\|b_i\|_1 \leqq 2b_{ii}$, the minimum again is $\lambda_i = b_{ii}$. Finally, if $\|b_i\|_1/2 < b_{ii} < \lambda_i$, the optimal value of the ith term is 1 for $\lambda_i = \infty$. Thus, if $\|b_i\|_1 \leqq 2b_{ii}$ for $i = 1, \dots, n$, a solution is given by the diagonal matrix built up by the eigenvalues of C_F. □

THEOREM 6. *Let A be a Hermitian positive definite matrix and $B = F_n A F_n^H$. A solution of*

$$\min_{\Lambda \text{ diagonal}} \|\Lambda^{1/2} - B\Lambda^{-1/2}\|_1$$

is given by

$$\lambda_i = \begin{cases} b_{ii} & \text{for } 3b_{ii} \geqq \|b_i\|_1 \\ \|b_i\|_1 - 2b_{ii} & \text{for } 3b_{ii} < \|b_i\|_1. \end{cases}$$

Proof. It holds that

$$\|\Lambda^{1/2} - B\Lambda^{-1/2}\|_1 = \max_{i=1}^{n} \left\{ \left|\sqrt{\lambda_i} - \frac{b_{ii}}{\sqrt{\lambda_i}}\right| + \frac{\|b_i\|_1 - b_{ii}}{\sqrt{\lambda_i}} \right\}.$$

For $\lambda_i \leqq b_{ii}$ or $\lambda_i > b_{ii} \geqq \|b_{ii}\|_1/3$, the ith term takes its minimum value for $\lambda_i = b_{ii}$. If $b_{ii} < \lambda_i$ and $\|b_i\|_1 > 3b_{ii}$, the ith term is minimal for $\lambda_i = \|b_i\|_1 - 2b_{ii}$. □

Let us denote by C_1 the circulant approximation that corresponds to the diagonal matrix of Theorem 6.

The last two theorems show that the Frobenius norm approximation C_F is often an optimal circulant approximation to a given matrix A relative to the norm $\|A\|_{F1} := \|F_n A F_n^H\|_1$. But in some cases we get circulant approximations that depend on $\|b_i\|_1$. Therefore, we have to consider $\|B\|_1$ to show that C_1 is well defined.

THEOREM 7. *Let T_n be a symmetric Toeplitz matrix connected with a positive function of the Wiener class, and $B = F_n T_n F_n^H$. If*

$$\sum_{r=1}^{n} |rt_r| \tag{8}$$

is bounded, then $\|B\|_1$ is bounded.

Proof. With (5) we get, for $k \neq j$,

$$\begin{aligned} b_{k,j} &= \frac{1}{n} \sum_{r=1}^{n-1} t_r w^{(1-k)r} \sum_{m=0}^{n-r-1} w^{(j-k)m} + \frac{1}{n} \sum_{r=1-n}^{-1} t_r w^{(1-k)r} \sum_{m=1-r}^{n} w^{(j-k)(m-1)} \\ &= \frac{1}{n(1-w^{j-k})} \sum_{r=1}^{n} \{t_r (w^{(1-k)r} - w^{(1-j)r} + w^{(j-1)r} - w^{(k-1)r})\} \\ &= \frac{w^{(k-j)/2}}{n \sin((k-j)\pi/n)} \sum_{r=1}^{n-1} t_r \left(2i \sin\left(\frac{2\pi(1-k)r}{n}\right) + 2i \sin\left(\frac{2\pi(j-1)r}{n}\right)\right) \\ &= \frac{4iw^{(k-j)/2}}{n \sin((k-j)\pi/n)} \sum_{r=1}^{n-1} t_r \sin\left(\frac{\pi r(j-k)}{n}\right) \cos\left(\frac{\pi r(k+j-2)}{n}\right). \end{aligned}$$

Then, for $j=1,\ldots,n, j\neq k$,

$$|b_{k,j}|\leqq\left(\frac{4}{n|\sin((j-k)\pi/n)|}\right)\sum_{r=1}^{n-1}|t_r\sin((j-k)r\pi/n)|.$$

Hence,

$$\sum_{j=1,j\neq k}^{n}|b_{k,j}|\leqq\frac{4}{n}\sum_{j=1,j\neq k}^{n}\sum_{r=1}^{n-1}|t_r|*\left|\frac{\sin(\pi r(j-k)/n)}{\sin(\pi(j-k)/n)}\right|$$
$$=4*\sum_{r=1}^{n-1}|t_r|\left\{\frac{1}{n}\sum_{j=1}^{n-1}\left|\frac{\sin(\pi rj/n)}{\sin(\pi j/n)}\right|\right.$$
$$=4*\sum_{r=1}^{n-1}|t_r|*f(r).$$

By induction it is easy to prove that $f(r)\leqq r$ for $r=1,\ldots,n$, and thus $\|B\|_1$ is bounded if (8) is fulfilled. □

It can be shown that $f(r)$ asymptotically satisfies the stronger inequality $f(r)\leqq c*\log(r)$ with c independent of n. The proof uses a partitioning of $f(r)$ for $\pi jr/n\in[k\pi-\pi/4, k\pi+\pi/4]$ and $\pi jr/n\in[k\pi+\pi/4, k\pi+3\pi/4]$, $k=0,1,\ldots,r$. Thus condition (8) in Theorem 7 can be replaced by the weaker condition that $t_r\log(r)$ is an l_1 sequence.

For the preconditioner C_1 we can not give any clustering analysis. Besides, the computation of $\|b_i\|_1$, $i=1,\ldots,n$, takes $O(n^2)$ operations, and thus is unattractive in practice. But, in many cases, $C_1=C_F$. Thus the main result of this section is that C_F is often an optimal approximation to a given matrix, not only in the Frobenius norm, but in other norms, as well.

3. Numerical examples. To test the different preconditioners, we display the spectra of the preconditioned matrices with $n=16$ for

(a) $t_0=2.007$, $t_1=-1$, $t_i=0$ for $i=2,\ldots,15$,

(b) $t_0=2.006$, $t_1=-1$, $t_i=0$ for $i=2,\ldots,15$,

(c) $t_0=0.6$, $t_i=\cos(i)/(i+1)$ for $i=1,\ldots,15$.

For these examples, $C_1\neq C_F$. C_B is not defined only for (b), and in that case, we examine C_{Bs} and C_{Bb}. See Figs. 1-3.

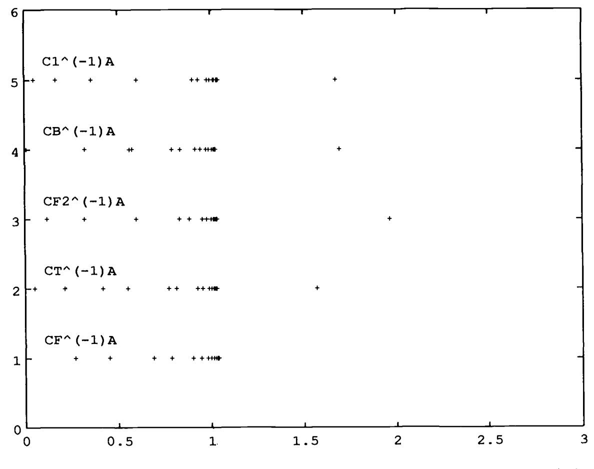


FIG. 1. *Spectra of the preconditioned systems for example (a).*

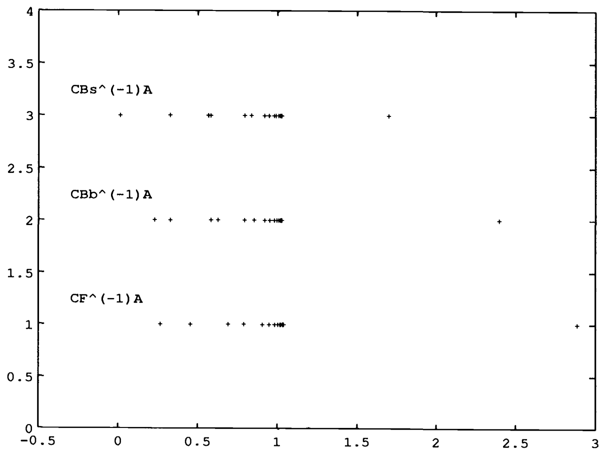


FIG. 2. *Spectra of the preconditioned systems for example* (*b*).

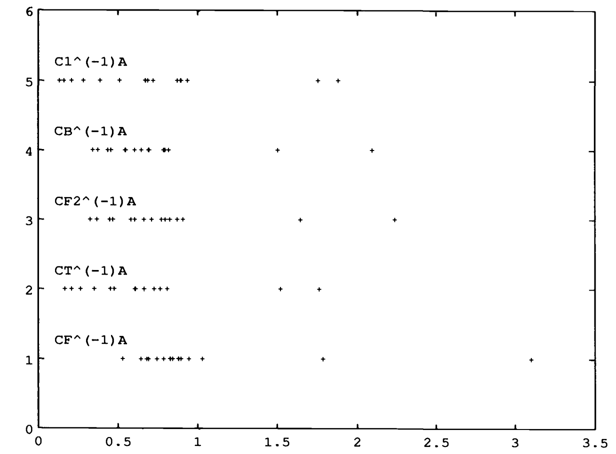


FIG. 3. *Spectra of the preconditioned systems for example* (*c*).

For example (b), Fig. 2 shows that C_{Bb} seems to be the better choice if C_B is not defined.

Next, we test the use of the PCG method for solving (4), $Dx = b$, with start vector $(1/b_{11}, \ldots, 1/b_{nn})^T$ for the examples (see Tables 1 and 2)

(d) $t(i) = 1/(i+1)^2, i = 0, \ldots, n-1$,

(e) $t_0 = 2, t_i = (1+\sqrt{-1})/(i+1)^{1.1}, i = 1, \ldots, n-1$.

TABLE 1
Error after the kth step for the PCG *method on* (4).

Example (d)	r_0	r_1	r_2
$n=8$	0.047	2.6×10^{-5}	1.2×10^{-7}
$n=16$	0.041	2.6×10^{-5}	5.7×10^{-8}
$n=32$	0.031	1.8×10^{-5}	5.6×10^{-9}
$n=64$	0.023	8.7×10^{-6}	9.0×10^{-10}
$n=128$	0.016	3.6×10^{-6}	2.2×10^{-10}

TABLE 2
Error after the kth step for the PCG *method on* (4).

Example (e)	r_0	r_1	r_2	r_3	r_4
$n=8$	0.096	0.0015	2.1×10^{-5}	2.4×10^{-7}	1.0×10^{-9}
$n=16$	0.13	0.0016	1.8×10^{-5}	2.6×10^{-7}	2.5×10^{-9}
$n=32$	0.13	0.0010	1.3×10^{-5}	1.8×10^{-7}	5.8×10^{-10}
$n=64$	0.12	4.5×10^{-4}	7.6×10^{-6}	7.1×10^{-8}	1.2×10^{-10}
$n=128$	0.095	1.8×10^{-4}	4.1×10^{-6}	2.0×10^{-8}	3.7×10^{-11}

Table 3 displays the number of iterations for the PCG method for solving $T_nx=b$ with example (e). The right-hand side is given by $b=(1,\dots,1)^T/\sqrt{n}$ and the initial guess is the zero vector. The algorithm stops if the Euclidean norm of the residuum is less than 10^{-6}.

Here, C_B is always defined, and C_1 differs from C_F for $n=64$ and $n=128$ on 2, respectively 8, positions.

TABLE 3
Number of iterations for different preconditioners in example (e).

n	C_F	C_{F2}	C_B	C_1
8	6	6	6	6
16	6	6	6	6
32	6	6	6	6
64	6	6	6	6
128	6	6	6	7

In most numerical experiments, the condition numbers of $C_B^{-1}T_n$ and $C_{F2}^{-1}T_n$ are smaller than that of $C_F^{-1}T_n$, but the clustering property of the spectrum of the preconditioned linear equation seems to be better for C_F.

REFERENCES

[1] R. CHAN, *The spectrum of a family of circulant preconditioned Toeplitz systems*, SIAM J. Numer. Anal., 26 (1989), pp. 503-506.

[2] ———, *Circulant preconditioners for Hermitian Toeplitz systems*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 542-550.

[3] R. CHAN, X.-Q. JIN, AND M.-C. YUENG, *The spectra of super optimal circulant preconditioned Toeplitz systems*, SIAM J. Numer. Anal., 28 (1991), pp. 871-879.

[4] R. CHAN AND G. STRANG, *Toeplitz equations by conjugate gradients with circulant preconditioner*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 104-119.

[5] R. H. CHAN AND T. F. CHAN, *Circulant Preconditioners for Elliptic Problems*, UCLA Computational and Applied Mathematics Report 90-32, Univ. of California, Los Angeles, CA; J. Numer. Linear Algebra Appl., to appear.

[6] T. CHAN, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 766-771.

[7] P. J. DAVIS, *Circulant Matrices*, John Wiley, New York, 1979.

[8] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[9] R. A. HORN AND C. R. JOHNSON, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, U.K., 1991.

[10] T. K. HUCKLE, *Circulant and skewcirculant matrices for solving Toeplitz matrix problems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 767-777.

[11] ———, *Krylovmethoden für normale Matrizen und Toeplitzmatrizen*, Habilitationsschrift, Universität Würzburg, Germany, July 1990.

[12] ———, *Circulant/Skewcirculant Matrices as Preconditioners for Hermitian Toeplitz Systems*, in Proc. IMACS Conf. on Iterative Methods in Linear Algebra, Brussels, April 1991.

[13] T. KU AND C. KUO, *Design and analysis of Toeplitz preconditioners*, Tech. Rep. 155, Signal and Image Processing Institute, Univ. of Southern California, Los Angeles, CA, May 1990; IEEE Trans. Acoust. Speech Signal Process., to appear.

[14] G. STRANG, *A proposal for Toeplitz matrix computations*, Stud. Appl. Math. 74 (1986), pp. 171-176.

[15] M. TISMENETSKY, *A decomposition of Toeplitz matrices and optimal circulant preconditioning*, Linear Algebra Appl., 154-156 (1991), pp. 105-121.

[16] E. TYRTYSHNIKOV, *Optimal and super optimal circulant preconditioners*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 459-473.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 542-569, May 1993

A PARALLEL ALGORITHM FOR THE NONSYMMETRIC EIGENVALUE PROBLEM*

JACK J. DONGARRA† AND MAJED SIDANI‡

Abstract. This paper describes a parallel algorithm for computing the eigenvalues and eigenvectors of a nonsymmetric matrix. The algorithm is based on a divide-and-conquer procedure and uses an iterative refinement technique.

Key words. eigenvalue problem, divide and conquer, parallel computing

AMS(MOS) subject classification. 65F15

1. Introduction. The algebraic eigenvalue problem is one of the fundamental problems in computational mathematics. It arises in many applications and therefore represents an important area of algorithmic research. The problem has received considerable attention, which has resulted in reliable methods [17]-[19]. However, it is reasonable to expect that calculations might be accelerated through the use of parallel algorithms. A fully parallel algorithm for the symmetric eigenvalue problem was recently proposed in [7]. This algorithm is based on a divide-and-conquer procedure outlined in [4]. The latter was based on work in [11] and [2]. The fundamental principle behind this algorithm is that the partitioning by rank-one tearing interlaces the eigenvalues of the modified problem with the eigenvalues of the original problem (the matrix is first reduced to tridiagonal form). This approach in turn enables rapid and accurate determination, in parallel, of the eigenvalues and the associated eigenvectors.

In this paper we propose a parallel algorithm for the solution of the nonsymmetric eigenvalue problem. The approach uses some of the features of the divide-and-conquer algorithm for the symmetric case mentioned earlier. In particular, the original problem is divided into two smaller and independent subproblems by a rank-one modification of the matrix. (We assume that the matrix has already been reduced to Hessenberg form, and that the rank-one modification removes a subdiagonal element.) Once the eigensystems of the smaller subproblems are known, it is possible to compute those of the original matrix. In the nonsymmetric case, the eigenvalues of the modified matrix do not interlace with those of the original matrix. Indeed, the eigenvalues can scatter anywhere in the complex plane.

In our algorithm for the nonsymmetric case, the eigensystem of the subproblem is used only to construct initial guesses for an iterative process which yields the desired eigensystem of the original problem. Under suitable conditions, iterative refinement or continuation can be used to find the eigenpairs of the original problem. We report here on our application of an iterative refinement approach based on Newton's method; we shall not pursue the continuation method in this paper. Work on the continuation approach has been reported by [14] and [15]. For other divide-and-conquer approaches, see [1] and [12].

* Received by the editors October 28, 1991; accepted for publication (in revised form) May 16, 1992. This work was supported in part by the National Science Foundation Science and Technology Center Cooperative Agreement CCR-8809615 and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract DE-AC05-84OR21400.

† Computer Science Department, University of Tennessee, Knoxville, Tennessee 37996-1301 (dongarra@cs.utk.edu).

‡ Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831 (sidani@cs.utk.edu).

In § 2 we describe an algorithm that uses an iterative refinement procedure based on Newton's method. Section 3 covers the deflation step required to overcome multiple convergence to a particular eigenvalue. In § 5 the convergence behavior of the new algorithm is discussed. In § 4 we discuss the case when the matrix or its rank-one modification has a defective system of eigenvectors. Section 7 estimates the amount of work the parallel algorithm requires and compares this to the standard techniques. Section 6 describes the parallel algorithm and the different parallel implementations of the new algorithm, and gives numerical results. Section 9 describes how our ideas extend to the generalized eigenvalue problem.

2. The algorithm. Given a matrix H, an eigenpair (x_0, λ_0) of H can be thought of as a solution to the polynomial system

$$(S)\begin{cases} Hx-\lambda x=0, \\ L(x)=1, \end{cases}$$

where $L(x)$ is a scalar equation. Here, we set $L(x)=e_s^T x$, where e_s is the sth unit vector (in practice, we choose s so as to normalize a known approximation to an eigenvector of H). Let

$$F_s(x,\lambda)=\begin{pmatrix} Hx-\lambda x \\ e_s^T x-1 \end{pmatrix}.$$

Then, finding an eigenpair of H reduces to finding a zero of F_s. In what follows, unless otherwise mentioned, H is assumed to be a real, unreduced (no zeros on the subdiagonal), upper-Hessenberg matrix of order n. This does not restrict the type of problems we want to solve, since if H has a zero on the subdiagonal then finding its eigenvalues reduces to finding those of the blocks on the diagonal. We note also by our assumption that H is unreduced, an eigenvalue of H can only have geometric multiplicity one: this is quite easy to see since the first $n-1$ columns of $H-\lambda I$ are linearly independent. We assume for now that H has a simple spectrum. We can write H as

$$H=\left(\begin{array}{c|c} H_{11} & H_{12} \\ \hline \alpha e_1^{(k)} e_k^{(k)T} & H_{22} \end{array}\right),$$

where H_{11} and H_{22} are upper-Hessenberg of dimensions $k\times k$ and $n-k\times n-k$, respectively; $\alpha=h_{k+1,k}$, and $e_i^{(k)}$ is the ith unit vector of length k.

Let $H_0=H-\alpha e_{k+1}^{(n)} e_k^{(n)T}$. Then

$$H_0=\left(\begin{array}{c|c} H_{11} & H_{12} \\ \hline 0 & H_{22} \end{array}\right) \quad \text{and} \quad \sigma(H_0)=\sigma(H_{11})\cup\sigma(H_{22})$$

(where $\sigma(M)$ is the spectrum of M). The algorithm can then be described as follows. We first find the k eigenpairs of H_{11} and the $n-k$ eigenpairs of H_{22} by some method. These eigenpairs are then used to construct initial approximations to the eigenpairs of H. If λ is an eigenvalue of H_{11} and x is the corresponding eigenvector, then λ is viewed as an approximate eigenvalue of H with the corresponding approximate eigenvector taken to be $\binom{x}{0}$, where $n-k$ zeros are appended to x. Note that $\binom{x}{0}$ is an exact eigenvector of H_0 corresponding to λ. On the other hand, if λ is an eigenvalue of H_{22} and x is the corresponding eigenvector, then $\binom{0}{x}$, where k zeros are prefixed to x, is taken as an approximate eigenvector of H corresponding to the approximate eigenvalue λ. If λ is not an eigenvalue of H_{11} then the last $n-k$ components of an exact eigenvector of H_0 corresponding to λ,

$$\begin{pmatrix} (H_{11}-\lambda)^{-1}H_{12}x \\ x \end{pmatrix},$$

are the components of x. However, if λ is an eigenvalue of H_{11}, and if its geometric multiplicity is one as an eigenvalue of H_0, then no eigenvector of H_0 will have the components of x as its trailing components. We have chosen to take $\binom{0}{x}$ as initial eigenvector instead of

$$\begin{pmatrix} (H_{11}-\lambda)^{-1}H_{12}x \\ x \end{pmatrix}$$

in order to avoid the additional computation involved in solving a linear system. The choice proved adequate in practice, in that there was no significant difference in the convergence behavior of the algorithm.

Newton's method comes into this problem in a rather "natural" way. Indeed, suppose that (x, λ) is an approximate eigenpair of H, $Hx \approx \lambda x$. Let us find a way to compute a correction (y, μ) to this approximate eigenpair. Clearly, (y, μ) should satisfy

$$H(x+y) = (\lambda+\mu)(x+y).$$

Rearranging the latter equation yields

$$(H-\lambda)y - \mu x = \lambda x - Hx + \mu y. \tag{1}$$

Now we ignore the second-order term μy, and we impose a normalization condition on x, say $x_s = 1$, where x_s is the sth component of x. If we also assume that the desired eigenvector should satisfy the same condition, then (y, μ) is the solution of

$$\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \mu \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}, \tag{2}$$

with $r = \lambda x - Hx$. But, this is the same equation that we obtain when a Newton iteration is applied to the function F_s to find a correction to (x, λ). We note here a result from [5], where the author studied an iterative refinement technique to compute the correction (y, μ) to (x, λ) from (1) (i.e., without ignoring the second-order term) and proved that in *exact* arithmetic, that method and Newton's method produce the same final iteration.

So far we have not attempted to answer the question of which subdiagonal entry will introduce the zero. We will use first-order perturbation theory in order to shed some light on this issue. Let $E = -\alpha e_{k+1}e_k^T$, where as above, $\alpha = h_{k+1,k}$ (note that $H + E = H_0$, introduced above). Then given a simple eigenpair (x, λ) of H, classical results from function theory [13, V.2, pp. 119–134] allow us to state that in a small neighborhood of zero, there exist differentiable $(x(\varepsilon), \lambda(\varepsilon))$ that satisfy

$$(H+\varepsilon E)x(\varepsilon) = \lambda(\varepsilon)x(\varepsilon),$$

for all ε in that neighborhood. Clearly, $x(0) = x$ and $\lambda(0) = \lambda$. Let y^H be the left eigenvetor of H corresponding to λ. Then differentiating both sides with respect to ε we have

$$HD_\varepsilon x(\varepsilon) + Ex(\varepsilon) + \varepsilon ED_\varepsilon x(\varepsilon) = D_\varepsilon\lambda(\varepsilon)x(\varepsilon) + \lambda(\varepsilon)D_\varepsilon x(\varepsilon),$$

where D_ε denotes the differentiation operator. Multiplying by y^H and setting $\varepsilon = 0$ we get

$$y^H Ex = D_\varepsilon \lambda(0) y^H x,$$

and therefore

$$|D_\varepsilon \lambda(0)| = \frac{|y^H Ex|}{|y^H x|} = \frac{|\alpha||y_{k+1}||x_k|}{|y^H x|}. \tag{3}$$

The quantities that vary with k in this expression are in the numerator. However, $|\alpha|$, $|y_{k+1}|$, and $|x_k|$ are not really independent of one another; indeed, if $\alpha = 0$, then at least one of y_{k+1} or x_k is zero. Hence for α small, we can expect one of y_{k+1} and x_k to be correspondingly small, since the components of the eigenvectors vary continuously. Therefore, we have found it sufficient in practice to look for the smallest subdiagonal entry in a prespecified range, and accept it as the subdiagonal entry (in that range) with respect to which the eigenvalues of the matrix are least sensitive, and set it equal to zero.

An outline of the algorithm follows.

ALGORITHM 2.1. Given an unreduced upper-Hessenberg matrix H, the following algorithm computes the eigensystems of two submatrices of H and uses them as initial guesses for starting Newton iterations for determining the eigensystem of H.

Determine subdiagonal element α where

$$H = \left(\begin{array}{c|c} H_{11} & H_{12} \\ \hline 0^\alpha & H_{22} \end{array}\right)$$

should be split; Determine initial guesses from eigensystems of the two diagonal blocks H_{11} and H_{22}; For each initial guess (λ_i, x_i) iterate until convergence:

$$\begin{pmatrix} H - \lambda_i I & -x_i \\ \hline e_s^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \mu \end{pmatrix} = \begin{pmatrix} r_i \\ 0 \end{pmatrix}; \quad \lambda_i \leftarrow \lambda_i + \mu; \quad x_i \leftarrow x_i + y;$$

end;
Check for duplicates and deflate if necessary;

More will be said about the last step, deflation, in § 3. In practice, the original matrix will be dense and we will need to reduce it to upper-Hessenberg form as a first step. This can be done in a stable fashion through a sequence of orthogonal similarity transformations, although elementary transformations can also be used with confidence, as in ELMHES [17].

A brief study of some sufficient conditions guaranteeing the convergence of our method will be touched upon in § 5. Now, assuming that the algorithm converges, it could happen that the same eigenpair of H is obtained more than once, i.e., starting from two (or more) distinct initial approximations, Newton's method converges to the same eigenpair of H. We have investigated methods to obtain further eigenpairs of H should this happen (see § 3).

We end this section with some implementational details. Our algorithm will accept (x, λ) as an eigenpair of H when $\|H_x - \lambda_x\| / \|x\| \, \|H\| < \text{tol}$, where tol is some specified tolerance of order ε, the machine unit roundoff. Under these conditions [9], (x, λ) is an exact eigenpair of a matrix obtained from H by a slight perturbation. Indeed,

$$\left(H+\frac{1}{x^H x} r x^H\right) x=\lambda x,$$

where $r=\lambda x-Hx$.

Starting from two complex conjugate initial approximations, Newton's method will converge to two complex conjugate zeros of F_s, or the same real zero. To prove this, it suffices to look at one iteration and show that when the current approximations are complex conjugate, Newton's method yields two complex conjugate corrections, or the same real correction. The system we must solve starting from (x, λ) is (2). For $(\bar{x}, \bar{\lambda})$ this becomes

$$\begin{pmatrix} H-\bar{\lambda} I & -\bar{x} \\ e_s^T & 0 \end{pmatrix}\begin{pmatrix} y' \\ \mu' \end{pmatrix}=\begin{pmatrix} \bar{r} \\ 0 \end{pmatrix}.$$

But this is the same linear system obtained when the conjugate of both sides in (2) is taken, since $\bar{H}=H$. Therefore, $y'=\bar{y}$ and $\mu'=\bar{\mu}$. This will allow significant savings in the computations.

Last, when starting from a real initial guess, only *real* corrections are computed. Therefore, the imaginary part of the eigenvalue should be perturbed if convergence ot a complex eigenpair is to be made possible. In practice, we have done this when real arithmetic does not lead to convergence after a prespecified number of corrections. A 2×2 real matrix whose eigenvalues are complex is a simple example of when perturbing the imaginary part of the eigenvalue is necessary.

3. Deflation.

When Newton's method is applied to solve the eigensystem of a matrix H, two distinct initial guesses may possibly converge to the same eigenpair of H. In fact, a naive implementation of the algorithm in § 2 may result in many eigenvalues being found multiple times and, consequently, some eigenvalues not being found at all. To avoid this unwanted situation, we included a deflation step in our algorithm that is designed to obtain further zeros.

Assume that when Algorithm 2.1 is applied to the $n\times n$ matrix H (we will assume for simplicity that it has no multiple eigenvalues) the eigenpair (x, λ) of H is obtained more than once, i.e., the algorithm converges to (x, λ) from several distinct initial guesses $(x_0^{(i)}, \lambda_0^{(i)})$, $i=1, \ldots, r$, $r>1$. There exist two classes of methods for finding the additional eigenpairs of H. The methods of one class produce an $(n-1)\times(n-1)$ matrix H' such that $\sigma(H')=\sigma(H)-\{\lambda\}$, and then Algorithm 2.1 is applied to H' starting from $r-1$ of these initial guesses. In this case, if the algorithm converges, then it will do so to eigenpairs different from (x, λ) since λ is no longer in the spectrum. Methods of this type will be discussed in §§ 3.1 and 3.3. The other class of methods will reapply Algorithm 2.1 to the original matrix H starting from $r-1$ of the initial guesses mentioned above, but will force convergence away from (x, λ) by ensuring, at all steps, that the current eigenvector forms a nonzero angle with x. A method of this type will be discussed in § 3.2.

A common drawback of all of these methods is that they tend to serialize the computation. However, it has been our experience that the need to deflate arises infrequently: less than 5 percent of the time in our tests.

3.1. Deflation using elementary transformations.

We now describe one possible deflating similarity transformation. We assume that H is an unreduced upper-Hessenberg matrix, λ is an eigenvalue of H, and x is the corresponding eigenvector.

Since we are assuming H to be upper-Hessenberg with no zeros on the subdiagonal, then $x_n \neq 0$, and the elementary transformation $M = [e_1, \dots, e_{n-1}, x]$ i.e.,

$$M = \begin{pmatrix} 1 & & & x_1 \\ & \ddots & & \vdots \\ & & 1 & x_{n-1} \\ 0 & & & x_n \end{pmatrix}, \tag{4}$$

is nonsingular. The inverse of this matrix is

$$M^{-1} = \begin{pmatrix} 1 & & & -x_1/x_n \\ & \ddots & & \vdots \\ & & 1 & -x_{n-1}/x_n \\ 0 & & & 1/x_n \end{pmatrix}. \tag{5}$$

It is easy to see that $M^{-1}x = e_n$, Now we let

$$\tilde{H} = M^{-1}HM. \tag{6}$$

It is easy to verify that $\tilde{H}$ is unreduced and upper-Hessenberg, and that the last column of $\tilde{H}$ is λe_n. Furthermore, the leading principal submatrix of order $n-1$ of $\tilde{H}$, which we will call H', is upper-Hessenberg, has the property that

$$\sigma(H') = \sigma(H) - \{\lambda\},$$

and differs from the leading $(n-1)\times(n-1)$ principal submatrix of H in the last column only. In fact, if we let h_{n-1} be the last column of the leading principal submatrix of H of order $n-1$, then it is straightforward to verify that the last column of H' is $h_{n-1} - h_{n,n-1}x'$, where

$$x' = \begin{pmatrix} x_1/x_n \\ \vdots \\ x_{n-1}/x_n \end{pmatrix}.$$

The strategy we have just described is given in [19] and applies regardless of whether the eigenpair (x, λ) is real or not. However, when (x, λ) is real we get a real matrix H'. In the case when (x, λ) is not real, the last column of H' alone is not real since, as we remarked earlier, the other columns are those of a leading principal submatrix of H. Hence the leading principal submatrix of H' of order $n-2$ is real. Also, in this case the complex conjugate of (x, λ), $(\bar{x}, \bar{\lambda})$ is an eigenpair of H, and therefore $\bar{\lambda}$ is an eigenvalue of $\tilde{H}$; a corresponding eigenvector is

$$M^{-1}\bar{x} = \begin{pmatrix} \bar{x}_1 - x_1(\bar{x}_n/x_n) \\ \vdots \\ \bar{x}_{n-1} - x_{n-1}(\bar{x}_n/x_n) \\ \bar{x}_n/x_n \end{pmatrix}.$$

Since $\sigma(H') = \sigma(H) - \{\lambda\}$, $\bar{\lambda}$ is an eigenvalue of H', and a corresponding eigenvector is the vector $\tilde{x}$ of length $n-1$, whose components are the first $n-1$ components of a scalar multiple of $M^{-1}\bar{x}$:

$$\tilde{x} = \begin{pmatrix} (\bar{x}_1 x_n - x_1\bar{x}_n)/i \\ \vdots \\ (\bar{x}_{n-1}x_n - x_{n-1}\bar{x}_n)/i \end{pmatrix},$$

where $i^2=-1$. This is due to the special structure of $\tilde{H}$. Now recall that H' has no zeros on the subdiagonal, and so we know that the last component of $\tilde{x}$ is nonzero. Note that $\tilde{x}$ is real. We can carry out a deflation that produces a matrix H'' of order $n-2$ with the property that

$$\sigma(H'')=\sigma(H')-\{\bar{\lambda}\};$$

in the same way, we obtained H' from H using the elementary transformation of order $n-1$:

$$M'=\begin{pmatrix} 1 & & \tilde{x}_1 \\ & \ddots & \vdots \\ & & \tilde{x}_{n-2} \\ 0 & & \tilde{x}_{n-1} \end{pmatrix}.$$

From a previous remark about this deflation strategy, we know that H'' differs from the leading principal submatrix of H' of order $n-2$ (which is real) in the last column only. The last column of H'' is $h'_{n-2}-h_{n-1,n-2}x''$, where h'_{n-2} is the last column of the leading principal submatrix of H', and x'' is the vector whose components are the first $n-2$ components of $\tilde{x}$. Since all of the quantities involved are real, H'' is real.

We remark here, as can be readily realized, that H' (or H'') is quite cheap to obtain in practice once an eigenpair of H is available. It requires $O(n)$ operations consisting of a vector normalization, a scalar-vector multiplication, and a vector-vector addition. However, the conditioning of the matrix M might raise concern. Indeed,

$$\operatorname{cond}_\infty(M)=\|M\|_\infty\|M^{-1}\|_\infty\approx\max(|x_i|)\max\left(\frac{|x_i|}{|x_n|}\right),$$

which can be large if $x_n \ll x_i$. Having noted this, it is clear that the ill-conditioning of M can be easily detected, and therefore one of the more stable (and costlier) methods that we introduce next and in the following sections can be used.

It is possible to prevent the ill-conditioning of M from bearing on the algorithm by avoiding a similarity transformation. More precisely, the eigenvalue problem we want to solve can be thought of as a generalized eigenvalue problem, $Hx=\lambda Bx$, with $B=I$. We want to find $\sigma(H)=\sigma(H,I)$. Now we know that given any nonsingular M and N,

$$\sigma(H,I)=\sigma(NHM,NM).$$

Given a particular eigenpair (x,λ), we would like to choose M and N in a way that solves the problem we set for ourselves at the beginning of this section, namely, we want to reduce the problem to one where λ is no longer in the spectrum. A closer look at the similarity transformation (6) reveals that its deflating property is due to the fact that $M^{-1}x=e_n$. But then M^{-1} is not the only matrix that can be used to accomplish this. In fact, the matrix N can be chosen to reduce x to a multiple of e_n: $N=DM^{-1}$, where D is the diagonal matrix with the entries

$$d_i=1,\quad \text{if } |x_i|/|x_n|\leqq 1,$$
$$d_i=-x_n/x_i\quad \text{if } |x_i|/|x_n|>1,$$

i.e., D is chosen so that all the entries in N are less than or equal to one. Then we have

$$(7)\qquad NHM=\left(\begin{array}{c|c} H' & 0 \\ \hline 0\quad\alpha & \gamma \end{array}\right),\qquad NM=\left(\begin{array}{c|c} D' & 0 \\ \hline 0 & \delta \end{array}\right),$$

with $\lambda = \lambda/\delta$. Also, $\sigma(H', D') = \sigma(H, I) - \{\lambda\}$, and therefore, by this transformation, λ has been "removed" from the spectrum. Working on the solution of a generalized eigenvalue problem from this point on will not generally cause any dramatic increase in the cost of the algorithm, mainly because D' is diagonal. A Newton step with this problem involves the following computation:

$$(8)\qquad \begin{pmatrix} H' - \lambda_i D' & -D'x_i \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \mu \end{pmatrix} = \begin{pmatrix} r_i \\ 0 \end{pmatrix}, \quad \lambda_i \leftarrow \lambda_i + \mu; \quad x_i \leftarrow x_i + y,$$

where $r_i = \lambda_i D' x_i - H x_i$. Clearly, the previous computation involves an $O(n)$ increase in the cost of one step: this comes from the multiplications by D'. The details on how (8) is derived are given in § 9. If λ is complex, then after deflating λ and its conjugate $\bar{\lambda}$, the resulting matrices H' and D' are generally complex. This is the major drawback of this method.

Finally, we mention another approach that can be of interest when the similarity transformation (6) involves a very ill conditioned M. This approach consists of interchanging two components of x and the corresponding columns and rows in H so that the last component of x is large enough. More precisely, let x_s be the largest component of x (in absolute value), and let P_{ns} be the matrix obtained from the identity matrix by permuting the nth and sth columns. Then $(P_{ns}x, \lambda)$ is an eigenpair of the matrix $P_{ns}HP_{ns}$, since

$$(P_{ns}HP_{ns})(P_{ns}x) = \lambda P_{ns}x.$$

Now scale the vector $P_{ns}x$ so that the last component is 1 and call that vector $\tilde{x}$. Then, as above, the elementary transformation

$$M = \begin{pmatrix} 1 & & \tilde{x}_1 \\ & \ddots & \vdots \\ & & \tilde{x}_{n-1} \\ 0 & & 1 \end{pmatrix}$$

can be used to deflate the matrix $P_{ns}HP_{ns}$. The fact that here $P_{ns}HP_{ns}$ is not upper-Hessenberg is of no consequence. In fact, we can make the following general statement: Given any matrix A of order n, and any eigenpair (x, λ) of A, then a matrix Q satisfying

$$Q^{-1}x = e_1 \quad \text{or} \quad Q^{-1}x = e_n$$

can be used to deflate A, in the sense that

$$Q^{-1}AQ = [\lambda e_1, B_1] \quad \text{or} \quad Q^{-1}AQ = [B_2, \lambda e_n],$$

respectively, where B_1 and B_2 are $n \times (n-1)$ matrices.

Having thus deflated the matrix $P_{ns}HP_{ns}$, the leading principal submatrix of order $n-1$ of

$$(9)\qquad M^{-1}P_{ns}HP_{ns}M$$

(call it H') has all the eigenvalues of H except λ (if λ is simple). However, H' is not generally upper-Hessenberg, and therefore will be reduced back to Hessenberg form before applying Newton's iterations; this is meant to save on the cost of factorizing the Jacobian when solving the linear systems arising at each step of Newton's iteration. Note that it is only the trailing diagonal submatrix of order $(n-s+1)\times(n-s+1)$ of

H' that needs to be reduced and that if $s = n - 1$ or $s = n$, then H' is upper-Hessenberg. Moreover, s need not be chosen so that x_s is the largest component of x (in absolute value). Indeed, since the size of the matrix to be reduced to upper-Hessenberg form increases when s approaches 1, it is more advantageous to choose the largest s for which the ratios x_i/x_s are moderate. We wish, therefore, to define a threshold t for the size of these ratios on the basis of which s will be determined.

Let $\tilde{H}$ be the computed form of the matrix in (9). In [19, Chap. 9], Wilkinson established that if $\|x\|_\infty \leqq 1$, then the eigenvalues of $\tilde{H}$ are the exact eigenvalues of a matrix $\tilde{H}'$ satisfying

$$\|H - \tilde{H}'\|_\infty \leqq \|r\|_\infty + (n-1)\varepsilon,$$

where r is the residual $\lambda x - Hx$ and ε is the machine epsilon. With no assumptions on the infinity norm of x, this inequality becomes

$$\|H - \tilde{H}'\|_\infty \leqq \|r\|_\infty + (n-1)\|x\|_\infty \varepsilon. \tag{10}$$

Since, in our algorithm, our computed eigenpairs have residuals on the order of $n\|H\|_\infty \varepsilon$, we propose that $t = \|H\|_\infty$.

In addition to destroying the structure of the matrix, this last method of deflation suffers from the fact that in the case when the eigenvalue to be deflated is nonreal, the resulting matrix H' is complex, and therefore will considerably increase the cost of finding subsequent eigenpairs if a Newton process is restarted from a real initial guess.

3.2. Deflation with help from the left eigenvector. The method we introduce now is different in spirit from the ones in the previous section, in that no attempt is made to modify the matrix.

Assume that (x, λ) is an exact eigenpair of H and that λ is simple $Hx = \lambda x$. No assumption is made about the remaining eigenvalues of H.

Let (x, X) be such that

$$(x, X)^{-1} H (x, X) = \begin{pmatrix} \lambda & 0 \\ 0 & J \end{pmatrix}$$

where the right-hand side is the Jordan canonical form of H. Now set

$$(x, X)^{-1} = \begin{pmatrix} y^H \\ Y \end{pmatrix}.$$

Then it is clear that y^H is a left eigenvector of H corresponding to λ and furthermore that

$$y^H X = 0.$$

This property can be used to modify Newton's method to avoid convergence to the eigenpair (x, λ) a second time. Indeed, given λ, we can compute the left eigenvector y corresponding to it, and use it to confine the current eigenvector to the range $R(X)$ of X. Therefore, we can expect to converge to an eigenvector linearly independent of x and hence corresponding to a different eigenpair (since (x, λ) was assumed to be simple). When (x, λ) has already been computed once, our algorithm for avoiding it then consists of the following major steps.

Compute the left eigenvector y^H corresponding to λ, with $\|y\|_2=1$; Given the current eigenpair (z, μ) compute the Newton correction from Algorithm 2.1; Let z' be the approximate eigenvector obtained after adding the Newton correction to z; choose the next eigenvector z'' as:

$$z''=(I-yy^H)z'.$$

In the last step we are just projecting z' onto $R(X)$. We note that when this algorithm is applied, it could happen (as with all the deflation methods we are describing) that we obtain another eigenpair of H, (x', λ'), that was already computed. Then the process must be restarted and z'' will be obtained from z' by a projection onto $R(X')$, where $X=(x', X')$, in order to avoid both (x, λ) and (x', λ'); this will require the computation of the left eigenvector corresponding to λ' as well.

It is obvious why the known eigenpair must be simple for the algorithm just outlined to work. If λ is multiple, then left eigenvectors are no longer necessarily orthogonal to X. In fact, the algorithm will be adversely affected if the eigenvalue λ is ill conditioned, i.e., if y^Hx is very small. Indeed, in this case, if the current eigenvector $z=\alpha x+Xv$, where v is a vector of length $n-1$, then

$$(I-yy^H)z=z-(y^Hz)y=\alpha x+Xv-(\alpha y^Hx)y\approx z,$$

showing that z is hardly modified by the projection and therefore suggesting that the algorithm will not necessarily prevent a second convergence to (x, λ).

The algorithm generalizes to the case when λ is multiple in the following way. Let V be a right invariant subspace corresponding to λ. Let (V, V_c) be such that

$$(V, V_c)^{-1}H(V, V_c)=\begin{pmatrix} J_\lambda & 0 \\ 0 & J \end{pmatrix},$$

where the right-hand side is again the Jordan canonical form of H. J_λ is the Jordan block corresponding to λ; since H is assumed to be unreduced, there can be only one such block. If we set

$$(V, V_c)^{-1}=\begin{pmatrix} U^H \\ W \end{pmatrix},$$

then clearly U^H is a left invariant subspace corresponding to λ, and furthermore,

$$U^HV_c=0.$$

This last property will allow U^H to be used in much the same way as the left eigenvector was used earlier. However, the practical usefulness of this method is restricted to the case when the eigenvalue λ is simple. Indeed, the problem of determining the invariant subspace associated with a multiple eigenvalue λ is an extremely difficult one and can be prohibitively expensive.

This method in its simplest form (using the left eigenvector) adds $O(n^2)$ work to the cost of finding one eigenpair distinct from (x, λ). This is the cost of computing the left eigenvector corresponding to λ; the cost of a single projection is $O(n)$.

3.3. Deflation with orthogonal transformations. We present now a very stable method for obtaining an upper-Hessenberg matrix H' with the property that it has all the eigenvalues of H except for λ [19]. We assume for now that the eigenpair (x, λ) is *exact.*

The strategy consists of $n-1$ major steps, where at each step a new zero is introduced in the last column of $H-\lambda I$ starting from the bottom. The configuration at the beginning of the rth step looks like

$$\tilde{H} = \begin{pmatrix} * & \cdots & * & * \\ * & & & \vdots \\ & \ddots & \vdots & * \\ & & * & 0 \end{pmatrix}$$

with zeros in the last $r-1$ components of the last column. The rth step then consists in a post-multiplication by G_r, where G_r is the (possibly complex) rotation in the plane $(n-r, n)$ designed to annihilate the $(n-r, n)$ element

$$G_r = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_r & & & & -s_r \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \\ & & & s_r & & & & c_r \end{pmatrix},$$

where $\|c_r\|^2 + \|s_r\|^2 = 1$. This post-multiplication will affect columns $n-r$ and n only, and therefore will not disturb zeros previously introduced in the last column. At the $(n-1)$ step, G_{n-1}, which is constructed to zero the $(2, n)$ element, will also zero the $(1, n)$ element. Indeed, assume that after the $(2, n)$ element has been zeroed we have some value α in the $(1, n)$ position; then we have

$$(H - \lambda I)G_1 \cdots G_{n-1} = \begin{pmatrix} * & \cdots & * & \alpha \\ * & & & \\ & \ddots & \vdots & \\ & & * & 0 \end{pmatrix}.$$

Now if we develop the determinant of this matrix by the last column we get

$$\det[(H - \lambda I)G_1 \cdots G_{n-1}] = \alpha b_1 \cdots b_{n-1},$$

where b_r is the rth subdiagonal element of $(H - \lambda I)G_1 \cdots G_{n-1}$:

$$\det[(H - \lambda I)G_1 \cdots G_{n-1}] = \det(H - \lambda I) = 0,$$

since $(\det(G_r) = 1)$ for $r = 1, \ldots, n-1$. But $b_r \neq 0$ for all r, since we have assumed that H had no zeros on the subdiagonal and since post-multiplication by a G_r can only increase the modulus of a subdiagonal element in $H - \lambda I$. Thus we must have $\alpha = 0$, and therefore, at the end of the $n-1$ steps just described, the last column is zero. Let us set $\mathcal{G} = G_1 \cdots G_{n-1}$ to simplify the notation. Then

$$\mathcal{G}^{-1} = \mathcal{G}^H = G_{n-1}^H \cdots G_1^H,$$

and it is straightforward to verify that the zeros of the last column of $(H - \lambda I)\mathcal{G}$ will be preserved when it is premultiplied by $\mathcal{G}^{-1}$, because the successive premultiplications by G_r^T, $r = 1, \ldots, n-1$, will preserve those zeros. Therefore, the last column of

$$\tilde{H} = \mathcal{G}^H(H - \lambda I)\mathcal{G} + \lambda I$$

is equal to λe_n. The eigenvector of $\tilde{H}$ corresponding to λ is e_n. Note, however, that $\tilde{H}$ is not upper-Hessenberg in this case. Indeed, nonzero elements will be introduced in the last row of $\tilde{H}$; in fact,

$$\tilde{H}=\begin{pmatrix} * & \cdots & & * & \\ * & & & & 0 \\ & \ddots & & \vdots & \\ & & * & * & \\ * & \cdots & & * & \lambda \end{pmatrix}.$$

This is not disturbing since if H' is the leading principal submatrix of $\tilde{H}$ of order $n-1$, then H' is upper-Hessenberg and $\sigma(H')=\sigma(H)-\{\lambda\}$.

If λ is a multiple eigenvalue of H of (algebraic) multiplicity m, then λ is an eigenvalue of H' of multiplicity $m-1$.

So far we have assumed that the eigenpair (x, λ) is exact. In practice, (x, λ) will only be approximate, in the sense that $Hx-\lambda x=r$ is of the order of the machine ε. In this case, roundoff errors will generally prevent G_{n-1} from annihilating the $(1, n)$ entry. In fact, the accuracy of the computed eigenvalue λ will come into play. If λ corresponds to an ill-conditioned eigenvalue of H, then it is possible that λ will be a rather poor approximation of the exact eigenvalue. As a consequence, the $(1, n)$ entry might not be negligible at all, and examples do exist where this is indeed the case [19]. A way around this difficulty is to construct the plane rotations $G_1, \ldots, G_{n-1}$ in a way to reduce the vector x to e_n and then apply the corresponding similarity transformation to H. Inequality (10) from § 3.1 holds when this is done (with obvious modification in the definition of H'). However, this will generally result in introducing nonzero entries below the subdiagonal of H and therefore H needs to be reduced to upper-Hessenberg form again. We refer the reader to [3] for an example of such an algorithm; the generalization of that algorithm to the case where the eigenvalue to be deflated is complex is straightforward.

In addition to the difficulty just mentioned, the deflation with plane rotations suffers from the fact that the deflated matrix will be complex if the eigenvalue to be deflated is complex. Indeed, it is unfortunately not true that when an eigenvalue and its complex conjugate are deflated by this method, the resulting matrix is real.

Example 3.1. When the two complex eigenvalues of the 4×4 upper-Hessenberg matrix

$$\begin{pmatrix} 0.2190 & -0.0756 & 0.6787 & -0.6391 \\ -0.9615 & 0.9032 & -0.4571 & 0.8804 \\ 0 & -0.3822 & 0.4526 & -0.0641 \\ 0 & 0 & -0.1069 & -0.0252 \end{pmatrix}$$

are deflated using plane rotations as just described, we obtain

$$\begin{pmatrix} 1.1593-0.0000i & 0.0000+1.0129i & 0.0000+0.0000i & 0.0000+0.0000i \\ 0.0000-0.3053i & 0.1740-0.0000i & 0.0000+0.0000i & 0.0000+0.0000i \\ -0.1534-0.3540i & 0.5717+0.0112i & 0.1082-0.4681i & 0.0000+0.0000i \\ -0.4640+0.1988i & 0.2187-0.3465i & 0.3964+0.0611i & 0.1082+0.4681i \end{pmatrix}.$$

After the deflation, we will be working with the upper 2×2 block of H' which is clearly complex.

3.4. Remarks on identifying duplicate eigenvalues. We have already remarked that the computed eigenpairs from our algorithm are the exact eigenpairs of a nearby matrix. Under those conditions [9],

$$|\lambda-\lambda_e| \leqq \|E\|/|s(\lambda_e)|,$$

where E is the error in the matrix, and $|s(\lambda_e)|$ is the condition number of the exact eigenvalue λ_e of the original matrix H. When λ_e is ill conditioned, we can expect unpredictably large errors (compared to the tolerated size of the residual) in the computed approximations to λ_e and therefore in the duplicates, if any. Identifying these duplicates becomes a rather daunting task: in particular, they will not be detected if their difference is compared to tol introduced earlier. Conversely, it can happen that under certain conditions the tolerated size of the residual will be much larger than the distance between certain exact eigenvalues and therefore between certain computed eigenvalues. In this case, it could happen that distinct computed eigenvalues will be declared as duplicates if their difference is compared to tol.

Example 3.2. We illustrate this last case with the following matrix:

$$H=\begin{pmatrix} 2\times 10^{-7} & 0 & 0 \\ 2 & 10^{-7} & 0 \\ 0 & 2 & 10^{10} \end{pmatrix}.$$

The tolerated size of the residuals for this matrix as chosen in our algorithm is $\text{tol} \approx \|H\|\varepsilon > 10^{-6}$. Therefore, if we decide to declare as duplicates those eigenvalues whose difference is less than tol, then the first two distinct eigenvalues of H will be declared as duplicates.

The problems we have just raised do not have easy solutions [10], and indeed, more research is needed here.

3.5. Conclusion regarding deflation techniques. As we pointed out earlier, the need to deflate arises less than 5 percent of the time in our tests. Our method of choice has been the method of deflation using elementary transformations introduced in § 3.1. This method is indeed the least expensive among all those we have discussed. Also, the resulting deflated matrix is in upper-Hessenberg form and is real when a pair of complex conjugate eigenvalues has been deflated.

4. Defective case. Our being a nonstationary iteration (the iterating map is not fixed), it is not easy to analyze the behavior of the successive approximations. We try, however, to address this problem in this section with a particular emphasis on the case when either the matrix H or the modified matrix H_0 is defective, i.e., when either one of these matrices does not have a complete set of eigenvectors. As we remarked earlier, an eigenvalue of H (with no zeros on the subdiagonal) can only have geometric multiplicity one, and there H is defective whenever it has a multiple eigenvalue. An eigenvalue of H_0, on the other hand, can have geometric multiplicity one or two. In what follows, n is the order of H.

The connection between Newton's method and inverse iteration is well known [16]. We derive this relationship in a way that motivates the subsequent analysis: We let J be the Jacobian of the map F_s at (x, λ) defined in § 2,

$$J=\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix},$$

and we assume that $x_s=1$. The order of the Jacobian is $n+1$. Then

$$J=J_0+e_{n+1}v^T$$

with

$$J_0 = \begin{pmatrix} H-\lambda I & -x \\ 0 & 1 \end{pmatrix}$$

and

$$v = e_s - e_{n+1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ -1 \end{pmatrix}.$$

Assume that J_0 is not singular; this is true if and only if λ is not an eigenvalue. Assume also that J is not singular. The correction to (x, λ) is computed in the following manner:

$$\begin{pmatrix} y \\ \mu \end{pmatrix} = J^{-1} \begin{pmatrix} r \\ 0 \end{pmatrix},$$

where $r = \lambda x - Hx$. Using the Sherman-Morrison formula [9], we can write J^{-1} as

$$J^{-1} = (J_0 + e_{n+1} v^T)^{-1} = J_0^{-1} - \frac{1}{1+v^T J_0^{-1} e_{n+1}} J_0^{-1} e_{n+1} v^T J_0^{-1}.$$

Therefore, letting $b(\begin{smallmatrix} r \\ 0 \end{smallmatrix})$, we have,

$$\begin{pmatrix} y \\ \mu \end{pmatrix} = J^{-1} b = J_0^{-1} b - \frac{v^T J_0^{-1} b}{1+v^T J_0^{-1} e_{n+1}} J_0^{-1} e_{n+1}. \tag{11}$$

It can be easily checked that

$$J_0^{-1} b = \begin{pmatrix} -x \\ 0 \end{pmatrix}$$

and that

$$J_0^{-1} e_{n+1} = \begin{pmatrix} \tilde{x} \\ 1 \end{pmatrix},$$

where $(H - \lambda I)\tilde{x} = x$. Now, if we let (x_1, λ_1) be the next eigenpair, we have from (11) and the subsequent equalities:

$$\begin{pmatrix} x_1 \\ \lambda_1 \end{pmatrix} = \begin{pmatrix} x \\ \lambda \end{pmatrix} + \begin{pmatrix} y \\ \mu \end{pmatrix} = \begin{pmatrix} x \\ \lambda \end{pmatrix} + \begin{pmatrix} -x \\ 0 \end{pmatrix} - \frac{v^T J_0^{-1} b}{1+v^T J_0^{-1} e_{n+1}} \begin{pmatrix} \tilde{x} \\ 1 \end{pmatrix}.$$

Furthermore,

$$\frac{v^T J_0^{-1} b}{1+v^T J_0^{-1} e_{n+1}} = \frac{-1}{\tilde{x}_s},$$

and so finally we see that our scheme reduces to the following: Given (x, λ), compute the next iterate (x_1, λ_1) via

$$(H - \lambda I)\tilde{x} = x, \quad x_1 = \frac{1}{\tilde{x}_s} \tilde{x}, \quad \lambda_1 = \lambda + \frac{1}{\tilde{x}_s}. \tag{12}$$

4.1. Case when H is defective. When the algorithm is expressed as in (12), we can readily see some of the difficulties that arise when the matrix H is defective or almost defective, which is generally more likely due to roundoff errors. These problems are similar to the kind of problems that we face with the application of inverse iteration. More precisely, assume that H is almost defective with $x_1, \dots, x_n$ as a complete set of eigenvectors. Let $\lambda_1, \dots, \lambda_l$ be a cluster of eigenvalues of H and suppose that the initial approximate eigenvalue λ corresponds to one of these. The eigenvectors $x_1, \dots, x_l$ corresponding to these eigenvalues are then almost linearly dependent. In general, the eigenvector corresponding to λ can be expected to converge to the space generated by $x_1, \dots, x_l$. As the eigenvalue λ approaches the cluster, however, continued corrections to the eigenvector cannot be expected to refine it. We refer the reader to the particularly lucid account in [16] for a justification of these claims. Solving as in inverse iteration (see INVIT [17]) is a possible way around this problem. Computing the residual with extended precision arithmetic is also an obvious approach, and has been successful in practice.

When approaching a singular solution, Newton's method loses its quadratic convergence rate. We will prove later (Theorem 5.1) that the Jacobian is singular at multiple eigenpairs. Therefore, we can expect slower convergence when multiple eigenpairs or almost multiple eigenpairs are the target: this is indicated in Fig. 1 by the large number of iterates separating the initial guesses from the converged values for an almost-defective matrix.

Recall the rate of change of λ that we derived in § 2:

$$|\lambda'(0)| = \frac{|\alpha| \, |y_{k+1}| \, |x_k|}{|y^H x|}.$$

We wish to caution against hastily drawing conclusions about the sensitivity of the eigenvalues of a defective matrix to our dividing process from this expression. Indeed, as an extreme case which will help to illustrate our point, the eigenvalues of a defective

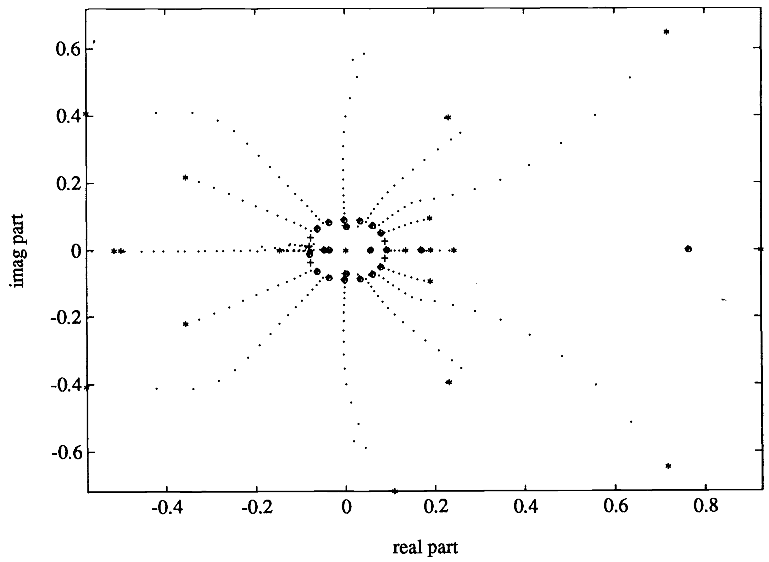


FIG. 1. *The behavior of the algorithm for an almost defective* 25×25 *matrix. The crosses are the eigenvalues of the original matrix; the stars are the initial guesses; the circles are the eigenvalues computed by our algorithm; the dots are the iterates arising in Newton's iterations.*

matrix can remain virtually unchanged after a zero has been introduced in the subdiagonal. An example is the 2×2 matrix

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

After the dividing process we have

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The starting eigenpairs are then $(\binom{1}{0}, 1)$ and $(\binom{0}{1}, 1)$. The second of these is, of course, the exact eigenpair. Now, even though the first starting eigenvector is orthogonal to the desired one, the equations arising during the first of Newton's iterations can be solved in such a way that the desired eigenvector is produced from the first step: zero pivots must be replaced by small numbers on the order of the machine unit roundoff (as is done in inverse iteration; see INVIT [17]).

Finally, we mention that the deflation process can contribute to the improvement of the condition of the eigenvalues. Indeed, if λ and λ' are pathologically close (and fairly distant from the rest of the eigenvalues), then by deflating λ, λ' will have a better condition number as an eigenvalue of the resulting matrix. Indeed, λ' is no longer part of a cluster.

4.2. Case when H_0 is defective. It can happen in this case (e.g., if H is nondefective) that the initial dividing process would leave us with a number of initial approximations that is smaller than n. In this case, random eigenvectors are used to complete the set of initial eigenvectors. Furthermore, whatever eigenpairs we have can be extremely poor approximations to the desired ones. An extreme situation is illustrated by the matrix

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

No matter where the zero is introduced on the subdiagonal, the resulting matrix has zero as its only eigenvalue, and we only have two initial approximations to the four distinct eigenpairs of H, namely,

$$\left(\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, 0\right), \quad \left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, 0\right),$$

if the zero is introduced in the $(3, 2)$ position.

Furthermore, the Jacobian is exactly singular at each of these initial approximations. Indeed, the Jacobian at

$$\left(\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, 0\right)$$

is

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix},$$

which is clearly of rank four, since the first and the last columns are linearly dependent. Similarly, it can be seen that the Jacobian at

$$\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, 0 \right)$$

is also singular. A remedy to this situation is to perturb the initial guess zero by a small amount, thereby making the Jacobian nonsingular. Indeed, this has been successful in practice. The problem in this case, as in most other similar cases, results from the particular structure of the matrix. In order to obtain further eigenvalues, we need to deflate the matrix each time a new eigenpair is computed which makes the algorithm almost serial.

4.3. Known failures. Some matrices of the structure mentioned at the end of § 4.2 (companion-like matrices), provided us with the only cases where the algorithm failed in practice to converge to the desired eigenpairs, i.e., failed to produce eigenpairs with small residuals after a fixed number of iterations. When these matrices were subjected to random orthogonal similarity transformations, however, and then reduced back to upper-Hessenberg form, the dividing process provided us with much better initial approximations and, indeed, the algorithm converged for all initial approximations. We are certainly not advocating this as a general viable scheme: we want to emphasize the fact that it is the structure of the matrix that caused the poor approximations and the failures, and not some inherent difficulty with the spectrum of these matrices.

5. Convergence. In § 2, we mentioned that computing an eigenpair of H reduces to computing a zero of

$$F_s(x, \lambda) = \begin{pmatrix} Hx - \lambda x \\ e_s^T x - 1 \end{pmatrix}.$$

The Jacobian of F_s at (x, λ) is

$$D_{(x,\lambda)}F_s(x, \lambda) = \begin{pmatrix} D_x(Hx - \lambda x) & D_\lambda(Hx - \lambda x) \\ D_x(e_s^T - 1) & D_\lambda(e_s^T - 1) \end{pmatrix} = \begin{pmatrix} H - \lambda I & -x \\ e_s^T & 0 \end{pmatrix},$$

where $D_x(F)$ denotes the derivative with respect to x of the function F. In this section, we give sufficient conditions for the convergence of our procedure. The result is a version of the Kantorovich theorem as it applies to our case.

THEOREM 5.1 (Wilkinson). *Assume that* (x, λ) *is an exact zero of* F_s. *Then*

$$\begin{pmatrix} H - \lambda I & -x \\ e_s^T & 0 \end{pmatrix}$$

is singular if and only if λ *is multiple.*

Proof. Let us assume first that λ is a multiple eigenvalue of H and that x is the corresponding (exact) eigenvector. Then there exists a nonzero vector y such that

$$y^H(H-\lambda I)=0 \quad \text{and} \quad y^H x=0;$$

y is simply a left eigenvector of H corresponding to λ. Thus $(y^H 0)$ is a nonzero vector and

$$(y^H 0)\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix}=0.$$

This proves that the Jacobian is singular.

Now, conversely, if

$$\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix}$$

is singular then there exists a nonzero vector $\binom{v}{\mu}$ such that

$$\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix}\begin{pmatrix} v \\ \mu \end{pmatrix}=0.$$

But then $v_s=0$ and $(H-\lambda I)v=\mu x$. If $\mu=0$, then v is nonzero since $\binom{v}{\mu}\neq 0$, and so v is an eigenvector that is linearly independent of x, since $x_s=1$ and $v_s=0$. Hence λ is a multiple eigenvalue in this case. If $\mu\neq 0$, then v is also nonzero; if it were zero then we would have

$$\mu x=(H-\lambda I)v=(H-\lambda I)0=0,$$

and hence $x=0$, contradicting our assumption on x. But $(H-\lambda I)^2 v=\mu(H-\lambda I)x=0$, and thus v is a nonzero vector that has grade 2. Therefore, λ is multiple in this case as well. $\square$

Remark. Since we are assuming that H is upper-Hessenberg and unreduced, an eigenvalue can only be nonderogatory, i.e., the associated eigenspace has dimension one.

The previous result applies to the Jacobian at a zero of F_s. We wish to know more about the Jacobian at those approximations arising during Newton's iteration before convergence to an eigenpair.

THEOREM 5.2. *Assume that* (x, λ) *is not a zero of* F_s. *Then*

$$\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix}$$

is singular if and only if at least one of the following is true:

(1) λ *is an eigenvalue of* H *and has an eigenvector whose sth component is zero.*

(2) x *belongs to the space generated by* $(c_1, \dots, c_{s-1}, c_{s+1}, \dots, c_n)$, *where* c_i *is the ith column of* $H-\lambda I$ (λ *may or may not be an eigenvalue*).

Proof. Assume first that (1) is true and let y be an eigenvector, $y_s=0$. Then $\binom{y}{0}$ is clearly in the null space of the Jacobian. If (2) holds and $x=(H-\lambda I)y$ with $y_s=0$, then $\binom{y}{1}$ is in the null space of the Jacobian.

Conversely, if the Jacobian is singular then there exists a vector $\binom{y}{\mu}$ such that,

$$\begin{pmatrix} H-\lambda I & -x \\ e_s^T & 0 \end{pmatrix}\begin{pmatrix} y \\ \mu \end{pmatrix}=0.$$

This implies that

$$(H-\lambda I)y=\mu x, \qquad e_s^T y=0,$$

and clearly $y_s=0$. We now consider two cases according to whether μ is zero or not. If μ is zero, then λ is an eigenvalue with corresponding eigenvector y; therefore, we are in case (1). If μ is not zero, then μx and hence x is in the range of $H-\lambda I$; therefore we are in case (2) since $y_s=0$. Note that λ may or may not be an eigenvalue in this last case. □

Remarks. The theorem tells us that more often than not, a singular Jacobian is an indication that the current eigenvalue has already converged, and this has been our experience indeed. The singularity of the Jacobian is also an indication of an ill-conditioned eigensystem. In fact, if we accept that the current eigenvector was moving in the "right" direction, then if it satisfies condition (2) of Theorem 5.2, we can say that the eigenvalue is acting like a multiple one since the eigenvector is also in the range of $(H-\lambda I)$. In practice, we have not encountered a situation where condition (1) applied. If we accept again that the eigenvector was moving in the right direction, then condition (1) implies that the eigenvalue is acting like an eigenvalue of geometric multiplicity more than one (since $x_s=1$), which is impossible since the matrix is unreduced.

The second derivative of F_s is a constant bilinear operator with norm equal to 2. In fact,

$$F_s''(x,\lambda)=\left(\begin{array}{cc|cc} 0 & -I & -I & 0 \\ 0 & 0 & 0 & 0 \end{array}\right).$$

Suppose now that our procedure is started with initial guess (x_0, λ_0). We now give sufficient conditions for the convergence of our procedure with the given initial guess. Let us first introduce

$$K=\|F_s'^{-1}(x_0,\lambda_0)\|$$

and

$$c_1=\|(x_1,\lambda_1)-(x_0,\lambda_0)\|,$$

where (x_1,λ_1) is the first iterate, i.e.,

$$(x_1,\lambda_1)^T=(x_0,\lambda_0)^T-F_s'^{-1}(x_0,\lambda_0)F_s(x_0,\lambda_0).$$

We call $(x_0,\lambda_0),\dots,(x_k,\lambda_k)$ the sequence of iterates produced by the algorithm. Now the classical Kantorovich theorem [6] gives the following result.

THEOREM 5.3. *If* $\beta_0=Kc_1<\frac{1}{4}$, *then the sequence* (x_k,λ_k) *converges quadratically starting from* (x_0,λ_0).

The process can be regarded as starting from any of the iterates (x_i,λ_i), and in fact it will often converge even when the conditions of the theorem are not satisfied at (x_0,λ_0). These conditions will then be met for some (x_k,λ_k) at which stage convergence becomes quadratic.

6. Parallel algorithms: Details and performance.

It is fairly straightforward to see from § 2 how to obtain a parallel algorithm. We discuss here certain details. The given, generally dense, matrix is first reduced to upper-Hessenberg form using a parallel blocked algorithm. Next comes the partitioning phase or "divide." This phase amounts to constructing a binary tree with each node representing a partition into two subproblems. It has been our practice to partition the matrix into a number of subproblems (at the lowest level) equal to the number of processors available on the target machine. Each of these problems may be spawned independently without fear of data conflicts; the computation at this level (the lowest) consists of calls to the EISPACK routine HQR2. The tree is then traversed in reverse order with Newton's method applied at each node, using the results from the children as initial approximations. Note here that the computation at a node does not have to wait for both children to complete in order to start: as a matter of fact, it can start as soon as one child has computed one eigenpair of a subproblem. In order to stress this point, we mention here that this is quite different from the situation in the symmetric case [7], where information from both children is needed before computations can start at the node. However, in practice, we have allowed computations to start at a node only after at least one child has completed; the need to check for duplicate eigenvalues and deflate if necessary has imposed further synchronization.

The algorithm has been implemented on computers with a shared memory and computers with distributed memory architectures.

6.1. Shared memory implementation. So far we have used SCHEDULE [8] to implement the algorithm on shared memory computers. SCHEDULE is a package of FORTRAN and C subroutines designed to aid in programming explicitly parallel algorithms for numerical calculations. An important part of this package is the provision of a mechanism for dynamically spawning processes even when such a capability is not present within the parallel language extensions provided for a given machine.

6.2. Distributed memory implementation. The current implementation on distributed memory machines requires that the matrix be stored on each processor. This obviously puts rather severe constraints on the size of problems that can be solved. With this implementation however, communication is needed only during the deflation phase. This implementation is best described through the contribution of a particular processor. Suppose that we have four processors at our disposal, $p_0, \ldots, p_3$, and that accordingly the matrix H has been divided into four subproblems, $H_0, \ldots, H_3$, that their common size is $n/4$, and that they occur in this order on the diagonal of the matrix. We describe now the contribution of p_2 by steps:

1. Call HQR2 to solve for the eigensystem of the matrix H_2.

2. Refine the output from step 1 to get $\frac{1}{2}$ the number of (i.e., $n/4$) eigenpairs of the matrix $H_{1/2}$

$$H_{1/2} = \left(\begin{array}{c|c} H_2 & B \\ \hline 0 \quad \alpha & H_3 \end{array}\right),$$

where $H_{1/2}$ is a submatrix of H.

3. Refine the output from step 2 to get $\frac{1}{4}$ the number of eigenpairs (i.e., $n/4$) of the matrix H.

As can be readily realized, no communication between processors is needed except for checking for eigenpairs to which convergence occurred from more than one initial approximation. For example, the eigenpairs of $H_{1/2}$ are generated on p_2 and p_3, and therefore we need to check for duplicate eigenpairs (on each processor separately, which requires no communication, and across both, which requires communication).

We are currently developing another implementation where blocks of columns of the matrix are stored on different processors. This storage scheme has been dictated to us by the need to call HQR2 at the lowest level. Indeed, to call the serial HQR2 requires that contiguous columns of the matrix reside on the same processor. Therefore, storage schemes more advantageous for linear system solving, such as wrap mapping of columns or rows, could not be used. The communication between processors for this second implementation is more intensive. Communication is needed when solving the linear systems arising in Newton's iterations as well as for the deflation phase. Also because of the storage scheme, we can expect the processors to become successively idle during the factorization of the Jacobian and the back solve for the correction. However, we have implemented an efficient scheme where the Jacobian is repartitioned by rows before the back solve takes place: the "reshuffling" of the submatrices takes place between processors that became idle after doing their part of the Gaussian elimination.

7. Work estimates. We assume in this section that we are given a real dense matrix A. Then, the first task in our algorithm is to reduce A to an upper-Hessenberg matrix H. This requires $10n^3/3$ operations (plus lower-order terms) if elementary transformations are used, since these matrices are to be accumulated.

The dividing process is now applied to H. Assume that $n=2^m r$ for some $m \geqq 1$, where r is not necessarily relatively prime to 2. In order to simplify the subsequent analysis we will assume that the dividing process produces submatrices of equal size, namely half the order of the original matrix. If we repeat the dividing process m times, we end up with 2^m matrices of size r, each of which is upper-Hessenberg. A submatrix of size $n/2^i$ obtained in the dividing process will be referred to as a matrix at the level i. The matrix H itself is at the level 0, and the r matrices referred to above are at the level m. Thus we have $m+1$ levels in total. Let $p'=2^m$ be the number of submatrices at the lowest level, and p the number of processors; we will assume that $p=2^q$, $q \leqq m$. The cost of finding the eigenpairs of a (Hessenberg) matrix at the lowest level (by the QR algorithm) is roughly 18 $(n/p')^3$. This figure is very approximate and assumes, among other things, that two QR steps are needed before a real or two complex conjugate eigenvalues are identified, and that the matrix has an equal number of real and complex eigenvalues [9]. Let $s_l=n/2^l$ be the size of one matrix at the level l. Let k_l be the average number of iterations needed to get one eigenpair of a matrix at the level l. k_l depends on the matrix and on its size, of course. We will make the following simplifying assumption, however: $k_l=k$, $l=0,\ldots,m$, i.e., we assume that the average number of steps required for convergence is the same at all levels. Our experience with the algorithm suggests that this is a realistic assumption as long as the size of the submatrices remains moderate. If the number of levels is increased to the point where we are left with small submatrices (less than 20×20, say), then k_l becomes significantly larger as l increases (for very small matrices it can be more than 10 on the average).

The cost of computing one correction at the level l is roughly $6s_l^2$. Indeed, one Newton iteration involves the solution of a linear system that is upper-Hessenberg, but for possible nonzeros in the last row, the order of this linear system is s_l+1. Therefore, two multipliers at most must be computed per column and, when updating the matrix, each of these will be used in $2(s_l+1-i)$ multiplications and additions, where i is the index of the column. The factorization of the Jacobian requires $2s_l^2$ operations in addition to $O(s_l)$ operations (including divisions and comparisons). The forward solve is $O(s_l)$ work and is negligible. The backsolve requires s_l^2 operations and computing the residual requires another s_l^2 operations. Therefore, for a real current approximation, one correction comes at the cost of $4s_l^2$. For a complex current eigenpair, this becomes $16s_l^2$ since the dominant operations are a roughly equal number of multiplications and additions. Since, as we indicated in § 2, only one of a conjugate pair of complex eigenpairs must be corrected, we can assume that $8s_l^2$ operations are required for correcting a complex eigenpair. Assuming again that the matrix has an equal number of real and complex eigenvalues, our estimate for the amount of work required for computing one correction at the level l becomes $6s_l^2$ operations.

We shall use n/p initial guesses to start n/p Newton processes on each processor. We now have the following work estimate on one processor, assuming that all processors share equally the cost of all the stages of the algorithm

$$W_p=\frac{10n^3}{3p}+18\left(\frac{p'}{p}\right)\left(\frac{n}{p'}\right)^3+\sum_{l=0}^{m-1}6\left(\frac{n}{p}\right)ks_l^2+2\frac{n^3}{p},$$

where $10n^3/3p$ is the processor's contribution to the reduction of the original matrix to upper-Hessenberg form; $18\ (p'/p)(n/p')^3$ is the cost of applying the QR algorithm to p'/p matrices of size n/p'; $\sum_{l=0}^{m-1} 6(n/p)ks_l^2$ is the cost of solving k linear systems with matrices of size s_l, repeating this for n/p initial guesses and for $l = 0, \ldots, m-1$; $2n^3/p$ is the processor's contribution to the computation of the eigenvectors of the original dense matrix A once those of H have been computed. The expression for the work can be rewritten as

$$W_p = \frac{n^3}{p}\left(\frac{16}{3} + \frac{18}{p'^2} + \frac{24k}{3}\left(1 - \left(\frac{1}{4}\right)^m\right)\right).$$

But $p' = 2^m$ and therefore $p'^2 = 4^m$, and hence

$$W_p = \frac{n^3}{p}\left(\frac{16}{3} + 8k + (18 - 8k)\frac{1}{4^m}\right), \tag{13}$$

where for ease of reference we redefine the various parameters: n is the order of the matrix, p is the number of processors, k is the average number of Newton iterations needed before an eigenpair is accepted, and m is the number of zeros introduced on the subdiagonal.

The cost of getting the eigenvalues and eigenvectors by the QR algorithm is $15n^3$ [9]. A reasonable value for k is 3; however, there are cases when k is 2 or less. There are also cases where k is larger than 3, mostly with matrices of small order or defective matrices.

It is easy to verify that for $k = 2$ and for $m = 1$ (one split) the model for the cost of the algorithm predicts that a sequential implementation of our algorithm is faster than EISPACK's Real General (RG). Our model predicts that a sequential implementation of our algorithm is slower than RG for problems where k equals or exceeds 2 (see Fig. 2). Here are some sample values of W_p, assuming that $k = 3$ and $p = p' = 2^m$, which means that the original problem is subdivided into a number of problems equal to the number of available processors.

$$p = 128 \rightarrow W_p = 0.2291n^3; \qquad p = 1024 \rightarrow W_p = 0.0286n^3.$$

Here we have assumed that the problem is large enough to allow the efficient use of that many processors. Note that the ratio of the work estimate from our model to the

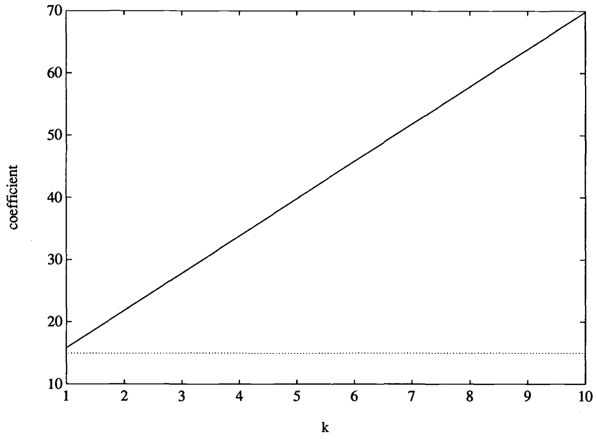


FIG. 2. *Variation of the coefficient of n^3 in work estimate model in terms of the number of steps k, with $m = 1$ and $p = 1$ (see expression for work).*

work estimate of QR is independent of n. This is due to the simplifying assumption on k made at the beginning of this section. That assumption has in effect "hidden" the dependency on n of the coefficient of n^3 in our model. Figures 2, 3, 4, and 5 show plots of the work estimate for various values of the parameters involved.

We note, finally, that the cost of our algorithm will increase when the matrix or the subproblems obtained in the divide process are ill conditioned (see § 4 and [12]). Furthermore, repeated deflations will also contribute to an increase in the cost.

8. Numerical results and performance. In this section we present the results of the implementation of the algorithm on a number of machines. The serial version of the code is available through NETLIB where it is called "nonsymdc."

The same algorithm has been run on the IBM RS/6000-550, the Alliant FX/8, the Intel iPSC/2, and the Intel iPSC/860. We compared our results to those of HQR2 from the EISPACK collection. We have used randomly generated upper-Hessenberg matrices in these tests, with entries uniformly distributed between -1 and 1; deflation

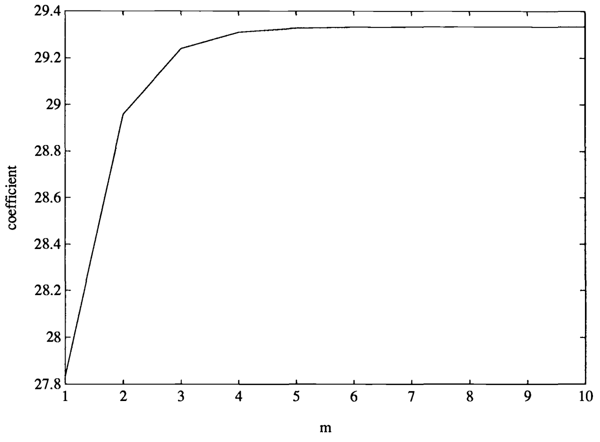


FIG. 3. *Variation of the coefficient of n^3 in work estimate model in terms of the number of splits m, with $k = 3$ and $p = 1$ (see expression for work).*

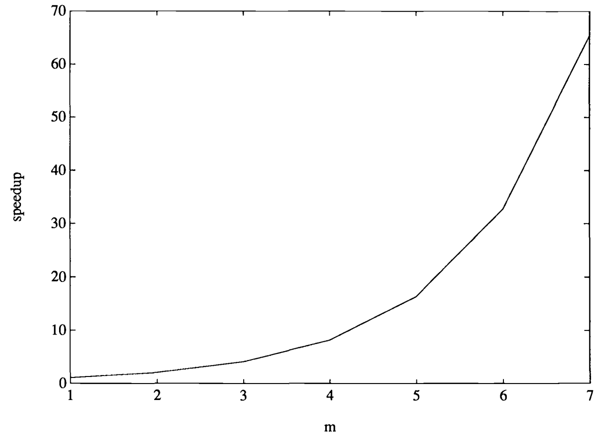


FIG. 4. *Predicted speedup over QR in terms of number of splits m, with $p = 2^m$ and $k = 3$ (see expression for work).*

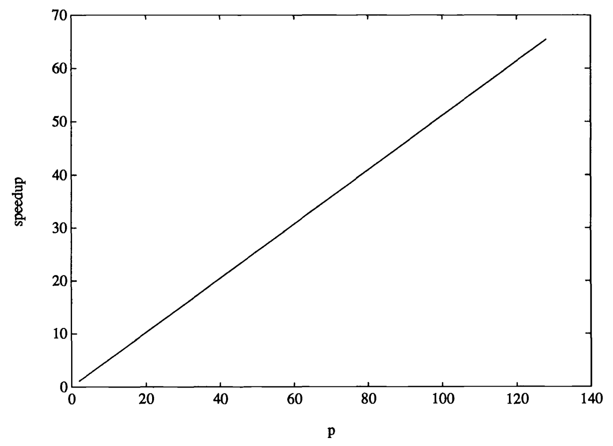


FIG. 5. *Predicted speedup over* QR *in terms of the number of processors* p, *with* $m=\log_2 p$ *and* $k=3$ (*see expression for work*).

was not performed. The storage requirement for our algorithm in a serial implementation is $4n^2+O(n)$, which is twice the storage requirement for a serial implementation of HQR [17].

Table 1 provides the results from the IBM RS/6000-550 implementation. The IBM RS/6000-550 is a single-processor computer with a RISC-based architecture. In the last column we have given the number of distinct eigenvalues that was computed by our algorithm (the test matrices had no multiple eigenvalues); some eigenvalues were found multiple times. Each matrix was divided into two subproblems, i.e., only one zero was introduced on the subdiagonal.

In an implementation on a shared memory machine, the storage allocated to the Jacobian (in serial mode) is multiplied by the number of processors used; this is meant to prevent concurrent write to the same memory locations. Table 2 provides some results from the Alliant FX/8 implementation. The Alliant FX/8 is a parallel machine with eight vector processors.

TABLE 1
Results on IBM RS/6000-550.

Order	HQR2	D&C	Ratio HQR2/D&C	Distinct λ
100	1.04	1.12	0.93	99
200	9.31	9.18	1.01	196
300	34.1	28.1	1.2	293
400	94.0	65.3	1.4	395
500	196	136	1.4	490
1000	1741	992	1.7	1000

TABLE 2
Results on Alliant FX/8.

Order	No. of procs.	Levels	Ratio HQR2/D&C
100	2	1	1.7
	4	2	2.4
	8	3	4.0

The results on the Alliant were generally disappointing. The storage scheme for the Jacobian that we used on that machine seems to have inhibited the compiler's vector optimizations. HQR2, running on a single processor, *was* vector optimized.

Table 3 provides some results from the Intel iPSC/2 implementation. The largest size used was dictated by the memory capacity of a single node.

Table 4 provides some results from the Intel iPSC/860 implementation. We observe here that the speedups realized by our algorithm over the QR algorithm do not remain linear for a large number of processors. This is due to the fact that our algorithm is much less efficient on small matrices, and we had to work with small matrices when the number of processors became large. For example, with a matrix of order 600 and using 64 processors, matrices of average size 20 had to be solved on each node at level 5.

TABLE 3
Results on iPSC/2.

Order	No. of procs.	Levels	Ratio HQR2/D&C
100	1	1	1.22
	2	1	2.2
	4	2	3.7
	8	3	6.0
	16	4	8.2
200	1	1	1.16
	2	1	2.2
	4	2	3.5
	8	3	5.2
	16	4	9.1
300	1	1	1.25
	2	1	2.2
	4	2	3.2
	8	3	6.3
	16	4	9.8
	32	5	13.2
	64	6	21.3

TABLE 4
Results on iPSC/860.

Order	No. of procs.	Levels	Ratio HQR2/D&C
100	1	1	1.15
	2	1	1.9
	4	2	3.3
	8	3	5.1
400	1	1	1.24
	2	1	2.4
	4	2	3.2
	8	3	6.0
	16	4	8.4
600	8	3	7.5
	16	4	13.5
	32	5	23
	64	6	32

9. The generalized eigenvalue problem. We show here how the ideas behind our algorithm can be used to solve the generalized eigenvalue problem.

9.1. Basic algorithm. Given an upper-Hessenberg matrix H and an upper-triangular matrix U,

$$H = \left(\begin{array}{c|c} H_{11} & H_{12} \\ \hline 0^{\alpha} & H_{22} \end{array}\right), \qquad U = \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array}\right),$$

we want to solve the generalized eigenvalue problem

$$Hx = \lambda Ux. \tag{14}$$

Without loss of generality, we can assume H to be unreduced and U to be nonsingular since otherwise the problem reduces to a smaller problem. Then our algorithm generalizes easily. Indeed, set

$$H_0 = H - \alpha e_{k+1} e_k^T,$$

and consider solutions of (or approximations of)

$$H_0 x = \lambda Ux, \tag{15}$$

as initial approximations to the sought eigenpairs. More precisely, solving (15) reduces to solving

$$H_{11} x = \lambda U_{11} x \quad \text{and} \quad H_{22} x = \lambda U_{22} x.$$

Let us denote these solutions by $(\tilde{x}_1, \lambda_1), \dots, (\tilde{x}_n, \lambda_n)$ (we have n solutions because of our assumption that U is nonsingular). These can be used to construct initial approximations $(x_1, \lambda_1), \dots, (x_n, \lambda_n)$ to the eigenpairs of the original generalized eigenproblem in much the same way we did in the case $U = I$. More precisely, we take

$$(x_i, \lambda_i) = \left(\begin{pmatrix} \tilde{x}_i \\ 0 \end{pmatrix}, \lambda_i \right)$$

if $\lambda_i \in \sigma(H_{11}, U_{11})$, and

$$(x_i, \lambda_i) = \left(\begin{pmatrix} 0 \\ \tilde{x}_i \end{pmatrix}, \lambda_i \right)$$

if $\lambda_i \in \sigma(H_{22}, U_{22})$ (appropriate number of zeros in each case).

Finally, solving the generalized eigenvalue problem (14) above is the same as solving the problem

$$Hx - \lambda Ux = 0, \quad L(x) = 1,$$

where $L(x)$ is a scalar equation, which we take to be a normalizing condition: $e_s^T x = 1$. Then for each initial eigenpair (x_i, λ_i), successive corrections can be computed via

$$\begin{pmatrix} H - \lambda_i U & -Ux_i \\ e_s^T & 0 \end{pmatrix} \begin{pmatrix} y \\ \mu \end{pmatrix} = \begin{pmatrix} r_i \\ 0 \end{pmatrix}, \quad \lambda_i \leftarrow \lambda_i + \mu; \quad x_i \leftarrow x_i + y,$$

where

$$r_i = \lambda_i U x_i - H x_i.$$

A possible stopping criterion for this scheme is the condition

$$\frac{\|Hx_i - \lambda_i U x_i\|}{\|x_i\|} \leqq f(n) \|H\| \, \|U\| \varepsilon,$$

where $f(n)$ is a modest function of n. It is easy to see that starting from two complex conjugate initial guesses the algorithm will compute complex conjugate corrections, therefore allowing savings similar to those in the case $U = I$.

9.2. Deflation in the generalized case. The method of deflation presented in § 3.1 generalizes to this case in the following manner. Let $Hx = \lambda Ux$ (of course, in practice, we only have an approximate eigenpair). Then it is true that if $w = Ux$ then $w_n \neq 0$. Indeed, if $w_n = 0$, then it can be shown that w is zero using the fact that H is unreduced. U being nonsingular implies that x is zero, which is not true.

Let

$$\begin{pmatrix} -v \\ 1 \end{pmatrix} = w/w_n$$

and

$$u = \begin{pmatrix} v \\ 1 \end{pmatrix}.$$

We know that $\sigma(H, U)$ is the same as $\sigma(NHM, NUM)$ for any nonsingular M and N. For our purposes, we take

$$M = [e_1, \dots, e_{n-1}, x]$$

and

$$N = [e_1, \dots, e_{n-1}, u].$$

Then it is easy to verify that we have

$$NHM = \left(\begin{array}{c|c} H' & 0 \\ \hline 0 \quad \alpha & \gamma \end{array}\right), \qquad NUM = \left(\begin{array}{c|c} U' & 0 \\ \hline 0 & \delta \end{array}\right), \tag{16}$$

where H' is upper-Hessenberg, U' is upper-triangular, and

$$\gamma/\delta = \lambda. \tag{17}$$

In fact, in this particular case we have $\gamma = \lambda$ and $\delta = 1$. Clearly,

$$\sigma(H', U') = \sigma(H, U) - \{\lambda\}.$$

Therefore, having "removed" λ from the spectrum we can get further eigenpairs. We note that, just as in the case $U = I$, H' and U' are very cheap to obtain once N has been determined. Indeed, H' differs from the $n-1 \times n-1$ principal submatrix of H in the last column only, whereas U' is the $n-1 \times n-1$ principal submatrix of U. The computation of N requires a matrix-vector multiply.

It is also easy to verify that deflating two consecutive complex conjugate eigenpairs results in real H' and U'.

The condition number of N might raise concern. We have

$$\text{cond}_\infty(N) \approx \max(|w_i|)^2.$$

It is therefore easy to detect an ill-conditioned N. We propose to handle this situation in the generalized case in the following manner. Let D be a diagonal matrix with its diagonal elements d_i defined by

$$d_i = 1, \qquad \text{if } \frac{|w_i|}{|w_n|} \leq 1,$$

$$d_i = -\frac{w_n}{w_i} \quad \text{if } \frac{|w_i|}{|w_n|} > 1.$$

Then, clearly,

$$\sigma(DNHM, DNUM) = \sigma(H, U),$$

since D is nonsingular. The multiplication by D cancels all the large entries in the last column of N. It is easy to see that (16) and (17) are satisfied when the premultiplication is done with DN instead of N. The disadvantage of having to premultiply by D is that after the deflation of two complex conjugate eigenpairs, the resulting H' and U' might still be complex.

REFERENCES

[1] L. ADAMS AND P. ARBENZ, *Towards a divide and conquer algorithm for the real nonsymmetric eigenvalue problem*, Tech. Rep. 165, Departement Informatik, ETH Zurich, Switzerland, September 1991.

[2] J. R. BUNCH, C. P. NIELSEN, AND D. C. SORENSEN, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31-48.

[3] P. A. BUSINGER, *Numerically stable deflation of Hessenberg and symmetric tridiagonal matrices*, BIT, 11 (1971), pp. 262-270.

[4] J. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195.

[5] J. DONGARRA, *Improving the accuracy of computed matrix eigenvalues*, Tech. Rep. ANL-80-84, Argonne National Lab., Argonne, IL, August 1980.

[6] J. DONGARRA, C. MOLER, AND J. WILKINSON, *Improving the accuracy of computed eigenvalues and eigenvectors*, SIAM J. Numer. Anal., 20 (1982), pp. 23-45.

[7] J. DONGARRA AND D. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. S139-S154.

[8] J. DONGARRA, D. SORENSEN, K. CONNOLLY, AND J. PATTERSON, *Programming methodology and performance issues for advanced computer architectures*, Parallel Comput., 8 (1988), pp. 41-58.

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

[10] G. GOLUB AND J. H. WILKINSON, *Ill-conditioned eigensystems and the computation of the Jordan canonical form*, SIAM Rev., 18 (1976), pp. 578-619.

[11] G. H. GOLUB, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318-334.

[12] L. E. JESSUP, *A case against a divide and conquer approach for the nonsymmetric eigenvalue problem*, Tech. Rep. ORNL/TM-11903, Oak Ridge National Lab., Oak Ridge, TN, 1991.

[13] K. KNOPP, *Theory of Functions*, Dover Publications, New York, 1945.

[14] T. Y. LI, Z. ZENG, AND L. CONG, *Homotopy-determinant algorithm for solving nonsymmetric eigenvalue problems*, Math. Comp., to appear.

[15] ———, *Solving eigenvalue problems of real nonsymmetric matrices with real homotopies*, SIAM J. Numer. Anal., 29 (1990), pp. 229-248.

[16] G. PETERS AND J. H. WILKINSON, *Inverse iteration, ill-conditioned equations and Newton's method*, SIAM Rev., 21 (1979), pp. 339-360.

[17] B. T. SMITH, J. M. BOYLE, J. J. DONGARRA, B. S. GARBOW, Y. IKEBE, V. C. KLEMA, and C. B. MOLER, *Matrix Eigensystem Routines-EISPACK Guide*, Lecture Notes in Comput. Sci., Vol. 6, Springer-Verlag, Berlin, New York, 1976.

[18] J. WILKINSON AND C. REINSCH, *Handbook for Automatic Computation: Volume* II—*Linear Algebra*, Springer-Verlag, Berlin, New York, 1971.

[19] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 570-584, May 1993

ON ROW RELAXATION METHODS FOR LARGE CONSTRAINED LEAST SQUARES PROBLEMS*

ACHIYA DAX†

Abstract. This paper addresses the question of how to construct a row relaxation method for solving large unstructured linear least squares problems, with or without linear constraints. The proposed approach combines the Herman-Lent-Hurwitz scheme for solving regularized least squares problems with the Lent-Censor-Hildreth method for solving linear constraints. However, numerical experiments show that the Herman-Lent-Hurwitz scheme has difficulty reaching a least squares solution. This difficulty is resolved by applying the Riley-Golub iterative improvement process.

Key words. row relaxation methods, large unstructured least squares problems, linear constraints, iterative improvement of regularized solutions

AMS(MOS) subject classification. 65K99

1. Introduction. This paper presents a row relaxation method for solving constrained least squares problems of the form

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|A\mathbf{x}-\mathbf{b}\|^2 \\ (1.1)\qquad &\text{subject to} && \mathbf{a}_i^T\mathbf{x} \geqq b_i \quad \text{for } i=m+1,\dots,s \\ &\text{and} && \mathbf{a}_i^T\mathbf{x} = b_i \quad \text{for } i=s+1,\dots,t, \end{aligned}$$

where $\|\ \|$ denotes the Euclidean norm. Here and henceforth, A is a real $m \times n$ matrix, $\mathbf{b}=(b_1,\dots,b_m)^T \in \mathbb{R}^m$, and $\mathbf{x}=(x_1,\dots,x_n)^T \in \mathbb{R}^n$ denotes the vector of unknowns. $\mathbf{a}_i$, $i=m+1,\dots,t$, are given vectors in $\mathbb{R}^m$, and b_i, $i=m+1,\dots,t$, are given real numbers. The rows of A are denoted by $\mathbf{a}_i^T$, $i=1,\dots,m$, and the $t \times n$ matrix whose rows are $\mathbf{a}_i^T$, $i=1,\dots,t$, is denoted by $\tilde{A}$.

The term *row relaxation method* refers to an iterative scheme whose basic iteration is composed of a sweep along the rows of $\tilde{A}$. This type of method is suitable for problems in which $\tilde{A}$ is large, sparse, and unstructured. In particular, it was proved to be a useful tool for handling large linear systems that arise in the field of image reconstruction from projections. In this field, it is possible to avoid storing $\tilde{A}$. Instead, the nonzero entries of the ith row are newly generated from the experimental data at each iteration. For this reason these methods are also called *row generation methods* or *row action methods* (see Censor [2]). Perhaps the most famous method of this kind is the method of Kaczmarz [11], which was rediscovered under the name ART (algebraic reconstruction technique) in the field of image reconstruction from projections (e.g., Gordon, Bender, and Herman [7] and Herman, Lent, and Rowland [8]). This scheme is aimed at solving a consistent linear system of the form

$$(1.2)\qquad A\mathbf{x}=\mathbf{b},$$

and its basic iteration is composed of m steps where the ith step, $i=1,\dots,m$, considers the ith equation. Let $\mathbf{x}$ denote the current estimate of the solution at the beginning of the ith step. Then the change in $\mathbf{x}$ during the ith step is

$$w\mathbf{a}_i(b_i-\mathbf{a}_i^T\mathbf{x})/\mathbf{a}_i^T\mathbf{a}_i,$$

* Received by the editors August 20, 1990; accepted for publication (in revised form) May 19, 1992.

† Hydrological Service, P.O. Box 6381, Jerusalem 91060, Israel.

where $0<w<2$ is a preassigned relaxation parameter. Note that when $w=1$, the current point is projected onto the hyperplane $\{\mathbf{u} \mid \mathbf{a}_i^T\mathbf{u}=b_i\}$. Another interpretation of this scheme lies in the following observations. The dual of the minimum norm problem

$$\text{(1.3)} \qquad \begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\mathbf{x}\|^2 \\ &\text{subject to} && A\mathbf{x}=\mathbf{b}, \end{aligned}$$

has the form

$$\text{(1.4)} \qquad \text{maximize} \quad \mathbf{b}^T\mathbf{y}-\tfrac{1}{2}\|A^T\mathbf{y}\|^2,$$

and if $\mathbf{y}^*$ solves the dual, then $A^T\mathbf{y}^*$ solves the primal. It is also easy to verify that $\mathbf{y}^*$ solves the dual if and only if it solves the system

$$\text{(1.5)} \qquad AA^T\mathbf{y}=\mathbf{b}.$$

Maximizing the dual objective function by changing one variable at a time results in the Gauss-Seidel method for solving (1.5). Moreover, let the sequence $\{\mathbf{y}_k\}$ be generated by applying the successive overrelaxation (SOR) method to solve this system, and let the sequence $\{\mathbf{x}_k\}$ be obtained by applying Kaczmarz's method, using $\mathbf{x}_0=A^T\mathbf{y}_0$ as the starting point. Then the equality

$$\text{(1.6)} \qquad \mathbf{x}_k=A^T\mathbf{y}_k$$

holds for all k (e.g., Björck and Elfving [1]). The relationship between Kaczmarz's method and the SOR method ensures that the sequence $\{\mathbf{x}_k\}$ always converges. If (1.2) is solvable then the limit point solves this system. Furthermore, if the starting point belongs to Range (A^T), then the limit point solves (1.3). However, when the system to be solved is inconsistent, the limit point is quite arbitrary and the method of Kaczmarz fails to provide a solution to the corresponding least squares problem

$$\text{(1.7)} \qquad \text{minimize} \quad \tfrac{1}{2}\|A\mathbf{x}-\mathbf{b}\|^2.$$

See Tanabe [14] and Dax [3] for a detailed explanation of this phenomenon. Example 1 in § 5 illustrates the inefficiency of Kaczmarz's method as a means for solving least squares problems. This raises the question of how to construct a modified Kaczmarz's method that is able to approach a least squares solution of an inconsistent system of linear equations.

One way to resolve this difficulty is to consider the regularized least squares problem

$$\text{(1.8)} \qquad \text{minimize} \quad \tfrac{1}{2}\varepsilon\|\mathbf{x}\|^2+\tfrac{1}{2}\|A\mathbf{x}-\mathbf{b}\|^2,$$

where ε is a positive constant. Let $\mathbf{x}_\varepsilon \in \mathbb{R}^n$ and $\mathbf{y}_\varepsilon \in \mathbb{R}^m$ denote the minimum norm solution of the extended linear system

$$\text{(1.9)} \qquad [A, \sqrt{\varepsilon}\, I]\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix}=\mathbf{b}.$$

Then $\mathbf{x}_\varepsilon$ provides the unique solution of (1.8). Moreover, since the extended system is solvable, it is possible to approach $\mathbf{x}_\varepsilon$ and $\mathbf{y}_\varepsilon$ by applying Kaczmarz's method on (1.9), which results in the method of Herman, Lent, and Hurwitz [9].

Tikhonov and others introduced the method of regularization as a means for improving the stability of ill-posed problems (e.g., Tikhonov [15] or Tikhonov and Arsenin [16]). In particular, it is a useful tool for handling unstable data-fitting problems. These are problems that result in an inconsistent system of linear equations $A\mathbf{x}=\mathbf{b}$, such that small changes in the data (i.e., the elements of A or $\mathbf{b}$) cause large changes in the minimum norm solution of (1.7). Recall that a point $\mathbf{x} \in \mathbb{R}^n$ solves (1.7) if and only if it solves the normal equations

$$A^T A\mathbf{x} = A^T \mathbf{b}. \tag{1.10}$$

Consequently, the minimum norm solution of (1.7) is defined as the unique vector $\mathbf{x}^*$, which solves the problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\mathbf{x}\|^2 \\ &\text{subject to} && A^T A\mathbf{x} = A^T\mathbf{b}. \end{aligned} \tag{1.11}$$

Assume that we have an a priori estimate $\mathbf{z}_0$ of the solution of (1.7). Then it is possible to stabilize the solution by adding the term $\frac{1}{2}\varepsilon\|\mathbf{x}-\mathbf{z}_0\|^2$ to our objective function. Also, there is no loss of generality in shifting the origin to be $\mathbf{z}_0$. This results in a problem of the form (1.8), which is usually referred to as *standard regularization* or *Tikhonov's regularization.*

The behavior of the regularized solution $\mathbf{x}_\varepsilon$ as a function of ε is easily inspected by applying the singular value decomposition (SVD) of A. Let $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ be orthogonal matrices such that

$$A = USV^T,$$

where $S \in \mathbb{R}^{m\times n}$ is a diagonal matrix. Define $p = \min\{m, n\}$ and let r denote the rank of A. Then, S is assumed to have the form

$$S = \operatorname{diag}\{\sigma_1, \dots, \sigma_p\},$$

where

$$\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_r > 0,$$

and

$$\sigma_i = 0 \quad \text{for } i = r+1, \dots, p.$$

The columns of U and V are denoted by $\mathbf{u}_1, \dots, \mathbf{u}_m$ and $\mathbf{v}_1, \dots, \mathbf{v}_n$, respectively. Now it is easy to verify that

$$\mathbf{x}^* = \sum_{i=1}^{r} (\mathbf{u}_i^T\mathbf{b}/\sigma_i)\mathbf{v}_i \tag{1.12}$$

and

$$\mathbf{x}_\varepsilon = \sum_{i=1}^{r} \alpha_i(\mathbf{u}_i^T\mathbf{b}/\sigma_i)\mathbf{v}_i, \tag{1.13}$$

where

$$\alpha_i = \sigma_i^2/(\sigma_i^2 + \varepsilon).$$

This presentation shows that instability is related to the presence of small nonzero singular values, while regularization dampens the effect of singular values, which are smaller than $\sqrt{\varepsilon}$. Further consequences of (1.13) are the limits

$$\lim_{\varepsilon\to 0^+} \mathbf{x}_\varepsilon = \mathbf{x}^*, \tag{1.14}$$

$$\lim_{\varepsilon\to \infty} \mathbf{x}_\varepsilon = \mathbf{0}, \tag{1.15}$$

and the observation that, as ε moves from zero to infinity, $\mathbf{x}_\varepsilon$ changes continuously from $\mathbf{x}^*$ to $\mathbf{0}$.

The limit (1.14) suggests that by choosing a small value of ε, the method of Herman et al. [9] is capable of providing a good estimate of $\mathbf{x}^*$. However, as our experiments illustrate, in practice, this idea does not always work. The reason for this discrepancy lies in the equivalence between Kaczmarz's method and the SOR method for solving (1.5). This relationship indicates that the method of Herman et al. [9] is equivalent to the SOR method for solving the system

$$(AA^T + \varepsilon I)\mathbf{y} = \mathbf{b}. \tag{1.16}$$

Hence, for small values of ε, the two methods actually coincide in the first iterations. The smaller the ε, the larger the number of iterations in which the two methods follow a similar path, and this may prevent the Herman-Lent-Hurwitz scheme from approaching $\mathbf{x}^*$ within a reasonable number of iterations.

The remedy proposed in this paper is based on the following idea. We have seen that one justification for solving the regularized problem is the existence of an initial estimate $\mathbf{z}_0$. However, when the regularized problem is solved we obtain a new estimate, $\mathbf{z}_1$. Hence, by shifting the origin to $\mathbf{z}_1$, we obtain an improved regularized problem and an improved solution $\mathbf{z}_2$. Similarly, $\mathbf{z}_2$ can be used to generate $\mathbf{z}_3$ and so forth. The basic iteration of this process is as follows: Given $\mathbf{z}_k$, we define

$$\mathbf{b}_k = \mathbf{b} - A\mathbf{z}_k \tag{1.17}$$

and solve the regularized least squares problem

$$\text{minimize} \quad \tfrac{1}{2}\varepsilon \|\mathbf{x}\|^2 + \tfrac{1}{2}\|A\mathbf{x} - \mathbf{b}_k\|^2. \tag{1.18}$$

Then the next point is defined as

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \mathbf{x}_k, \tag{1.19}$$

where $\mathbf{x}_k$ denotes the unique solution of (1.18).

The above iterative improvement process is due to Riley [13] and Golub [6], who considered it in the context of factorization methods for dense linear systems. An equivalent way to write this process is

$$\mathbf{z}_{k+1} = H\mathbf{z}_k + \mathbf{h}, \tag{1.20}$$

where

$$\mathbf{h} = (A^TA + \varepsilon I)^{-1}A^T\mathbf{b}$$

and

$$H = \varepsilon(A^TA + \varepsilon I)^{-1}.$$

The last equality enables us to express H in the form

$$H = VDV^T,$$

where D is a diagonal matrix,

$$D = \text{diag}\{d_1, \ldots, d_n\},$$

such that

$$d_i = \varepsilon/(\sigma_i^2 + \varepsilon) \quad \text{for } i = 1, \ldots, r,$$

and

$$d_i = 1 \quad \text{for } i = r+1, \ldots, n.$$

Golub [6] has used these relations to show that if $\mathbf{z}_0 = \mathbf{0}$, then

$$\mathbf{z}_k = \sum_{i=1}^{r} (1 - d_i^k)(\mathbf{u}_i^T\mathbf{b}/\sigma_i)\mathbf{v}_i, \tag{1.21}$$

and the sequence $\{\mathbf{z}_k\}$ converges to $\mathbf{x}^*$, the minimum norm solution of (1.7). Yet the fact that (1.20) describes a linear stationary iterative process enables us to extend this result in the following way. Since H is convergent and symmetric, and the system

$$(I-H)\mathbf{x}=\mathbf{h} \tag{1.22}$$

is solvable, the sequence $\{\mathbf{z}_k\}$ always converges, and the limit point solves this system. Moreover, if $\mathbf{z}_0 \in \text{Range }(I-H)$, then the limit point is the minimum norm solution of this system (see Dax [3, Corollary 6]). Now the definitions of $\mathbf{h}$ and H imply that $\mathbf{x}$ solves (1.22) if and only if it solves (1.10), and that Range $(I-H) =$ Range (A^T). In other words, the sequence $\{\mathbf{z}_k\}$ converges to a point that solves (1.7), and if $\mathbf{z}_0 \in$ Range (A^T), it converges to $\mathbf{x}^*$, the minimum norm solution of (1.7).

Let $\hat{\mathbf{z}}$ denote the limit point of the sequence $\{\mathbf{z}_k\}$. The optimality conditions of (1.18) imply the equality

$$\varepsilon \mathbf{x}_k = -A^T(A\mathbf{z}_{k+1}-\mathbf{b}). \tag{1.23}$$

This relation shows that $\mathbf{z}_{k+1}-\hat{\mathbf{z}} \in$ Range (A^T) and yields the inequality

$$\varepsilon \|\mathbf{x}_k\| = \|A^T(A\mathbf{z}_{k+1}-\mathbf{b})\| = \|A^TA(\mathbf{z}_{k+1}-\hat{z})\| \geqq \sigma_r^2 \|\mathbf{z}_{k+1}-\hat{\mathbf{z}}\|. \tag{1.24}$$

Hence, if the iterative improvement process stops at a point $\mathbf{z}_{k+1}$ that satisfies

$$\|\mathbf{x}_k\| \leqq \delta \|\mathbf{z}_{k+1}\|, \tag{1.25}$$

where δ is a small positive constant, we obtain the inequality

$$\|\mathbf{z}_{k+1}-\hat{\mathbf{z}}\| / \|\mathbf{z}_{k+1}\| \leqq \varepsilon\delta/\sigma_t^2, \tag{1.26}$$

which can be used to estimate the relative error in the computation of $\hat{\mathbf{z}}$. In particular, when $\mathbf{z}_0 = \mathbf{0}$, we conclude from (1.21) that

$$\|\mathbf{z}_k\| < \|\mathbf{z}_{k+1}\| < \|\mathbf{x}^*\|, \tag{1.27}$$

and (1.26) leads to

$$\|\mathbf{z}_{k+1}-\mathbf{x}^*\| / \|\mathbf{x}^*\| \leqq \varepsilon\delta/\sigma_r^2. \tag{1.28}$$

Let us return to the question of how to construct a row relaxation method that is able to find a least squares solution of an inconsistent linear system. If ε is not too small, then the SOR method for solving (1.6) enjoys a fast rate of convergence, and the Herman-Lent-Hurwitz scheme is likely to provide a good estimate of $\mathbf{x}_k$, the solution of (1.18), within a few iterations. This opens the way for practical implementation of the iterative improvement process as a row relaxation method. However, as equality (1.21) shows, a large value of ε slows down the convergence of the iterative improvement process. The success of the combined method depends, therefore, on a proper choice of ε that ensures a reasonable rate of convergence for both methods (see Table 1).

In the following sections we show that similar ideas can be used to construct a row relaxation method for solving constrained least squares problems. The properties of regularized problems of this kind are briefly discussed in § 2, while § 3 considers the corresponding iterative improvement process. Analysis of the constrained case requires a different approach, since now the SVD of A is not quite so helpful. The basic tool for handling large systems of linear inequalities is the method of Lent and Censor [12], which can be viewed as a row relaxation version of the method of Hildreth [10]. This scheme is discussed in § 4, in which we propose a modification that enables it to solve regularized constrained least squares problems. Combining the modified

Lent–Censor–Hildreth scheme with the iterative improvement process results in a row relaxation method that is able to solve large problems of the form (1.1). Finally, in § 5, we provide numerical experiments that demonstrate the feasibility of our ideas.

2. Regularization of constrained least squares problems. The regularized form of (1.1) is defined as

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\varepsilon\|\mathbf{x}\|^2+\tfrac{1}{2}\|A\mathbf{x}-\mathbf{b}\|^2 \\ &\text{subject to} && \mathbf{a}_i^T\mathbf{x}\geqq b_i \quad \text{for } i=m+1,\dots,s, \\ &\text{and} && \mathbf{a}_i^T\mathbf{x}=b_i \quad \text{for } i=s+1,\dots,t, \end{aligned} \tag{2.1}$$

where, as before, ε is a given positive constant. In this section we show that the basic properties of (2.1) are similar to those of (1.1). It is assumed here that the feasible region is not empty, and this ensures that both (1.1) and (2.1) have solutions. The minimum norm solution of (1.1) is the unique vector $\mathbf{x}^*$ that solves the problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\mathbf{x}\|^2 \\ &\text{subject to} && \|A\mathbf{x}-\mathbf{b}\|^2\leqq\|A\mathbf{z}-\mathbf{b}\|^2, \\ & && \mathbf{a}_i^T\mathbf{x}\geqq b_i \quad \text{for } i=m+1,\dots,s, \\ &\text{and} && \mathbf{a}_i^T\mathbf{x}=b_i \quad \text{for } i=s+1,\dots,t, \end{aligned} \tag{2.2}$$

where $\mathbf{z}$ denotes a solution of (1.1). We shall start by answering the question: what happens when ε tends to zero?

THEOREM 1. *Let $\{\varepsilon_k\}$ be a sequence of positive numbers such that*

$$\lim_{k\to\infty}\varepsilon_k=0, \tag{2.3}$$

and let $\mathbf{x}_k$ denote the solution of (2.1) *that corresponds to $\varepsilon=\varepsilon_k$. Then*

$$\lim_{k\to\infty}\mathbf{x}_k=\mathbf{x}^*. \tag{2.4}$$

Proof. The definitions of $\mathbf{x}^*$ and $\mathbf{x}_k$ imply

$$\|A\mathbf{x}^*-\mathbf{b}\|^2\leqq\|A\mathbf{x}_k-\mathbf{b}\|^2$$

and

$$\tfrac{1}{2}\varepsilon_k\|\mathbf{x}_k\|^2+\tfrac{1}{2}\|A\mathbf{x}_k-\mathbf{b}\|^2\leqq\tfrac{1}{2}\varepsilon_k\|\mathbf{x}^*\|^2+\tfrac{1}{2}\|A\mathbf{x}^*-\mathbf{b}\|^2.$$

Hence, by combining these inequalities we obtain

$$\tfrac{1}{2}\varepsilon_k\|\mathbf{x}_k\|^2+\tfrac{1}{2}\|A\mathbf{x}_k-b\|^2\leqq\tfrac{1}{2}\varepsilon_k\|\mathbf{x}^*\|^2+\tfrac{1}{2}\|A\mathbf{x}_k-\mathbf{b}\|^2$$

and

$$\|\mathbf{x}_k\|\leqq\|\mathbf{x}^*\|. \tag{2.5}$$

That is, the sequence $\{\mathbf{x}_k\}$ is bounded and has at least one cluster point, say $\hat{\mathbf{x}}$. Therefore, since (2.2) has a unique solution, it is sufficient to show that $\hat{\mathbf{x}}=\mathbf{x}^*$.

Let $\{\mathbf{x}_j\}$ be a subsequence of $\{\mathbf{x}_k\}$ such that

$$\lim_{j\to\infty}\mathbf{x}_j=\hat{\mathbf{x}}. \tag{2.6}$$

Then, since each $\mathbf{x}_j$ is a feasible point, $\hat{\mathbf{x}}$ is feasible. Similarly, the inequality (2.5) implies

$$\|\hat{\mathbf{x}}\| \leqq \|\mathbf{x}^*\|. \tag{2.7}$$

Hence, the proof is concluded by showing that $\hat{\mathbf{x}}$ solves (1.1).

Let $\hat{\mathbf{x}}$ be a feasible point, and let $Y(\hat{\mathbf{x}})$ denote the set of all the points

$$\mathbf{y} = (y_{m+1}, \dots, y_s, y_{s+1}, \dots, y_t)^T \in \mathbb{R}^{t-m},$$

such that

$$y_i \geqq 0 \quad \text{and} \quad y_i(\mathbf{a}_i^T\hat{\mathbf{x}} - b_i) = 0 \quad \text{for } i = m+1, \dots, s.$$

Then $\hat{\mathbf{x}}$ solves (1.1) if and only if there exists $\hat{\mathbf{y}} \in Y(\hat{\mathbf{x}})$ such that

$$A^T(A\hat{x} - \mathbf{b}) = \hat{A}^T\hat{\mathbf{y}}, \tag{2.8}$$

where $\hat{A}$ is the $(t-m) \times n$ matrix whose rows are $\mathbf{a}_i^T$, $i = m+1, \dots, t$. (These are the well-known Kuhn-Tucker optimality conditions.) Similarly, the optimality conditions of (2.1) indicate that for each $\mathbf{x}_j$ there exists a point $\mathbf{y}_j \in Y(\mathbf{x}_j)$ such that

$$\varepsilon_j\mathbf{x}_j + A^T(A\mathbf{x}_j - \mathbf{b}) = \hat{A}^T\mathbf{y}_j. \tag{2.9}$$

It is also easy to verify that the limit (2.6) implies the existence of an index $\hat{j}$ such that

$$\mathbf{y}_j \in Y(\hat{\mathbf{x}}) \quad \text{for } j \geqq \hat{j}. \tag{2.10}$$

A further use of (2.6) gives

$$\lim_{j\to\infty} \varepsilon_j\mathbf{x}_j = \mathbf{0} \tag{2.11}$$

and

$$\lim_{j\to\infty} \hat{A}^T\mathbf{y}_j = A^T(A\hat{\mathbf{x}} - \mathbf{b}). \tag{2.12}$$

Therefore, by the well-known projection theorem, there exists a point $\hat{\mathbf{y}} \in Y(\hat{\mathbf{x}})$ that solves the least squares problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\|\hat{A}^T\mathbf{y} - A^T(A\hat{\mathbf{x}} - \mathbf{b})\|^2 \\ &\text{subject to} && \mathbf{y} \in Y(\hat{\mathbf{x}}), \end{aligned} \tag{2.13}$$

while (2.10) and (2.12) imply that $\hat{\mathbf{y}}$ solves (2.8). □

The arguments we have used in the above discussion are easily modified to establish the following assertions.

THEOREM 2. *Let $\{\varepsilon_k\}$ be a sequence of numbers such that*

$$\lim_{k\to\infty} \varepsilon_k = \infty, \tag{2.14}$$

and let $\mathbf{x}_k$ denote the solution of (2.1) *that corresponds to $\varepsilon = \varepsilon_k$. Then*

$$\lim_{k\to\infty} \mathbf{x}_k = \tilde{\mathbf{x}}, \tag{2.15}$$

where $\tilde{\mathbf{x}}$ denotes the unique solution of the least distance problem

$$\begin{aligned} &\textit{minimize} && \tfrac{1}{2}\|\mathbf{x}\|^2 \\ &\textit{subject to} && \mathbf{a}_i^T\mathbf{x} \geqq b_i \quad \textit{for } i = m+1, \dots, s, \\ &\textit{and} && \mathbf{a}_i^T\mathbf{x} = b_i \quad \textit{for } i = s+1, \dots, t. \end{aligned} \tag{2.16}$$

THEOREM 3. *Let $\{\varepsilon_k\}$ be a sequence of positive numbers such that*

$$\lim_{k\to\infty} \varepsilon_k = \varepsilon,$$

and let $\mathbf{x}_k$ denote the solution of (2.1) *that corresponds to $\varepsilon = \varepsilon_k$. Then*

$$\lim_{k\to\infty} \mathbf{x}_k = \mathbf{x}_\varepsilon,$$

where $\mathbf{x}_\varepsilon$ denotes the solution of (2.1).

Summarizing our results, we conclude that as ε moves from infinity to zero, $\mathbf{x}_\varepsilon$ changes continuously from $\tilde{x}$ to $\mathbf{x}^*$.

3. Iterative improvement under constraints. This section considers the iterative improvement process as a tool for solving constrained least squares problems of the form (1.1). The basic iteration of the modified process is carried out as follows. Given $\mathbf{z}_k$, we define

$$b_i^{(k)} = b_i - \mathbf{a}_i^T\mathbf{z}_k, \qquad i = 1, \dots, t,$$

and calculate $\mathbf{x}_k$, the unique solution of the problem

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\varepsilon\|\mathbf{x}\|^2 + \tfrac{1}{2}\|A\mathbf{x} - \mathbf{b}_k\|^2 \\ &\text{subject to} && \mathbf{a}_i^T\mathbf{x} \geqq b_i^{(k)} \quad \text{for } i = m+1, \dots, s, \\ &\text{and} && \mathbf{a}_i^T\mathbf{x} = b_i^{(k)} \quad \text{for } i = s+1, \dots, t, \end{aligned} \tag{3.1}$$

where

$$\mathbf{b}_k = \mathbf{b} - A\mathbf{z}_k = (b_1^{(k)}, \dots, b_m^{(k)})^T \in \mathbb{R}^m. \tag{3.2}$$

Then the next point is defined as

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \mathbf{x}_k. \tag{3.3}$$

The above definitions imply the inequality

$$\|A\mathbf{z}_k - \mathbf{b}\|^2 \geqq \tfrac{1}{2}\varepsilon\|\mathbf{x}_k\|^2 + \tfrac{1}{2}\|A\mathbf{z}_{k+1} - \mathbf{b}\|^2,$$

which shows that the sequence $\{\|A\mathbf{z}_k - \mathbf{b}\|^2\}$ is monotonic decreasing, and

$$\lim_{k\to\infty} \mathbf{x}_k = 0. \tag{3.4}$$

Another justification for applying this process lies in the following observation.

THEOREM 4. *Assume that the sequence $\{\mathbf{z}_k\}$ has a cluster point $\hat{\mathbf{x}}$. Then $\hat{\mathbf{x}}$ solves* (1.1).

Proof. Let $\{\mathbf{z}_j\}$ be a subsequence of $\{\mathbf{z}_k\}$ such that

$$\lim_{j\to\infty} \mathbf{z}_j = \hat{\mathbf{x}}. \tag{3.5}$$

Then, since each $\mathbf{z}_j$ is a feasible point, $\hat{\mathbf{x}}$ is feasible. On the other hand, the optimality conditions of (3.1) imply that for each k there exists a vector $\mathbf{y}_k \in Y(\mathbf{z}_{k+1})$ such that

$$\varepsilon\mathbf{x}_k + A^T(A\mathbf{z}_{k+1} - \mathbf{b}) = \hat{A}^T\mathbf{y}_k. \tag{3.6}$$

Hence, by combining (3.4), (3.5), and (3.6), we obtain the limit

$$\lim_{j\to\infty} \hat{A}^T\mathbf{y}_j = A^T(A\hat{\mathbf{x}} - \mathbf{b}), \tag{3.7}$$

where $\{\mathbf{y}_j\}$ denotes the subsequence of $\{\mathbf{y}_k\}$ that corresponds to $\{\mathbf{z}_j\}$. Another consequence of (3.5) is the existence of an index $\hat{j}$ such that $\mathbf{y}_j \in Y(\hat{\mathbf{x}})$ for all $j \geqq \hat{j}$. Therefore, the least squares problem (2.13) has a solution $\hat{\mathbf{y}} \in Y(\hat{\mathbf{x}})$ that satisfies (2.8), which proves that $\hat{\mathbf{x}}$ solves (1.1). $\square$

COROLLARY. *If* (1.1) *has a unique solution, then the sequence* $\{\mathbf{z}_k\}$ *converges to this point.*

4. A row relaxation method for regularized constrained least squares problems.

The original method of Hildreth [10] is designed to solve certain types of convex quadratic programming problems. Later, Lent and Censor [12] have suggested a row relaxation version of Hildreth's method that is aimed at solving least distance problems of the form

$$\text{(4.1)}\qquad \begin{aligned} &\text{minimize} \quad \tfrac{1}{2}\|\mathbf{x}\|^2 \\ &\text{subject to} \quad \mathbf{a}_i^T\mathbf{x} \geqq b_i \quad \text{for } i = 1, \ldots, m. \end{aligned}$$

The basic iteration of the Lent-Censor scheme is similar to that of Kaczmarz's method, but here, in addition to the primal variables, the algorithm stores and updates dual variables. Let $\mathbf{x}$ and $\mathbf{y} = (y_1, \ldots, y_m)^T \in \mathbb{R}^m$ denote the current estimates of the variables at the beginning of the ith step of the kth iteration, $i = 1, \ldots, m$, $k = 1, 2, \ldots$. Then the *ith* step is carried out as follows:

(a) Calculate $\delta = -\min\{y_i, w(\mathbf{a}_i^T\mathbf{x} - b_i)/\mathbf{a}_i^T\mathbf{a}_i\}$.

(b) Set $\mathbf{x} := \mathbf{x} + \delta\mathbf{a}_i$ and $y_i := y_i + \delta$.

The symbol := denotes arithmetic assignment. That is, $y_i := y_i + \delta$ means "set the new value of y_i to be $y_i + \delta$." The starting points must satisfy

$$\text{(4.2)}\qquad \mathbf{x} = A^T\mathbf{y}$$

and

$$\text{(4.3)}\qquad \mathbf{y} \geqq \mathbf{0},$$

and these relations are kept throughout the iterative process. The motivation behind the previous scheme lies in the observation that the dual of (4.1) has the form

$$\text{(4.4)}\qquad \begin{aligned} &\text{minimize} \quad \tfrac{1}{2}\|A^T\mathbf{y}\|^2 - \mathbf{b}^T\mathbf{y} \\ &\text{subject to} \quad \mathbf{y} \geqq \mathbf{0}, \end{aligned}$$

and if $\hat{\mathbf{y}}$ solves the dual, then the vector $\hat{\mathbf{x}} = A^T\hat{\mathbf{y}}$ provides the unique solution of (4.1). Note that when $w = 1$, the value of δ provides the solution of the one-parameter problem

$$\text{(4.5)}\qquad \begin{aligned} &\text{minimize} \quad f(\theta) = \tfrac{1}{2}\|A^T(\mathbf{y} + \theta\mathbf{e}_i)\|^2 - \mathbf{b}^T(\mathbf{y} + \theta\mathbf{e}_i) \\ &\text{subject to} \quad y_i + \theta \geqq 0, \end{aligned}$$

where $\mathbf{e}_i$ denotes the ith column of the $m \times m$ unit matrix. It is also easy to verify that $-w(\mathbf{a}_i^T\mathbf{x} - b_i)/\mathbf{a}_i^T\mathbf{a}_i$ is the change of y_i during the ith step of the SOR method for solving the system

$$\text{(4.6)}\qquad AA^T\mathbf{y} = \mathbf{b}.$$

Therefore, the updating rule of the dual variables can be viewed as a modified SOR method in which the variables are restricted to stay nonnegative. The discussion in Lent and Censor [12] provides an interesting geometric interpretation of this method. Moreover, let $\mathbf{x}_k$ denote the current estimate of the primal variables at the end of the kth iteration and assume that the feasible region is not empty. Then it is shown there that the sequence $\{\mathbf{x}_k\}$ converges, and the limit point solves (4.1).

The adaptation of the Lent-Censor scheme to handle equality constraints is done in the following way. Consider, for example, a linear equality of the form $\mathbf{a}_i^T\mathbf{x} = b_i$. Then by writing this equality as two inequalities, $\mathbf{a}_i^T\mathbf{x} \geqq b_i$ and $-\mathbf{a}_i^T\mathbf{x} \geqq -b_i$, we obtain that the corresponding dual variable y_i is not restricted to stay nonnegative. Hence, in this case, there is no need to store and update y_i, and the ith step of the Lent-Censor scheme coincides with that of Kaczmarz's method:

(a) Calculate $\delta = -w(\mathbf{a}_i^T\mathbf{x} - b_i)/\mathbf{a}_i^T\mathbf{a}_i$.

(b) Set $\mathbf{x} := \mathbf{x} + \delta\mathbf{a}_i$.

In other words, the Lent-Censor scheme is essentially an extended version of Kaczmarz's method, which enables us to handle inequality constraints. The observation that the Hildreth method can be extended to include linear equalities, in the manner described above, is also made in Elfving [5], where it is shown that this extension does not ruin the convergence properties.

The implementation of the Lent-Censor scheme to solve regularized problems of the form (2.1) is based on the following observation, which is a straightforward extension of Lemma 2 of Herman et al. [9]. Let the vectors $\tilde{\mathbf{x}} \in \mathbb{R}^n$ and $\tilde{\mathbf{y}} \in \mathbb{R}^m$ provide the minimum norm solution of the system

$$(4.7a) \qquad (A, \sqrt{\varepsilon}\, I)\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \mathbf{b},$$

$$(4.7b) \qquad \mathbf{a}_i^T\mathbf{x} \geqq b_i \quad \text{for } i = m+1, \dots, s,$$

$$(4.7c) \qquad \mathbf{a}_i^T\mathbf{x} = b_i \quad \text{for } i = s+1, \dots, t.$$

Then $\tilde{\mathbf{x}}$ solves (2.1).

The minimum norm solution of (4.7) can be obtained by applying the Lent-Censor scheme to solve this system. The basic iteration of the resulting algorithm is as follows. For $i = 1, \dots, m$ the ith step coincides with Kaczmarz's method for solving (4.7a):

(a) Calculate $\delta = -w(\mathbf{a}_i^T\mathbf{x} + \sqrt{\varepsilon}\, y_i - b_i)/(\mathbf{a}_i^T\mathbf{a}_i + \varepsilon)$.

(b) Set $\mathbf{x} := \mathbf{x} + \delta\mathbf{a}_i$ and $y_i := y_i + \delta\sqrt{\varepsilon}$.

For $i = m+1, \dots, s$ the ith step coincides with the Lent-Censor scheme for solving (4.7b):

(a) Calculate $\delta = -\min\{y_i, w(\mathbf{a}_i^T\mathbf{x} - b_i)/\mathbf{a}_i^T\mathbf{a}_i\}$.

(b) Set $\mathbf{x} := \mathbf{x} + \delta\mathbf{a}_i$ and $y_i := y_i + \delta$.

For $i = s+1, \dots, t$ the ith step coincides with Kaczmarz's method for solving (4.7c):

(a) Calculate $\delta = -w(\mathbf{a}_i^T\mathbf{x} - b_i)/\mathbf{a}_i^T\mathbf{a}_i$.

(b) Set $\mathbf{x} := \mathbf{x} + \delta\mathbf{a}_i$.

The algorithm may start with any pair of points $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^t$ that satisfy

$$(4.8) \qquad \mathbf{x} = \left(\sum_{i=1}^{m} \mathbf{a}_i y_i\right)\Big/\sqrt{\varepsilon} + \sum_{i=m+1}^{t} \mathbf{a}_i y_i$$

and

$$(4.9) \qquad y_i \geqq 0 \quad \text{for } i = m+1, \dots, s,$$

and these relations hold throughout the iterative process. Note that the algorithm does not store and updates the values of $y_{s+1}, \dots, y_t$. Let $\mathbf{x}_k$ denote the current estimate of the solution at the end of the kth iterations. Then the convergence properties of the Lent-Censor scheme indicate that the sequence $\{\mathbf{x}_k\}$ converges, and the limit point is the unique solution of (2.1).

The above scheme enables us to implement the iterative improvement process that has been described in the previous section. In practice there is no need to calculate the exact solution of (3.1), and $\mathbf{z}_{k+1}$ is obtained from $\mathbf{z}_k$ by applying a few iterations of the row relaxation scheme. The justification for this shortcut lies in the observation that Theorem 4 remains valid when the requirement that $\mathbf{x}_k$ solve (3.1) is replaced by the following three conditions.

(a) $GA\mathbf{z}_k-\mathbf{b}\|^2 \geqq \frac{1}{2}\varepsilon\|\mathbf{x}_k\|^2+\frac{1}{2}\|A\mathbf{z}_{k+1}-\mathbf{b}\|^2$.

(b) $\lim_{k\to\infty} \mathbf{d}_k=0$, where $\mathbf{d}_k$ denotes the distance between $\mathbf{z}_k$ and the feasible region.

(c) For each k, there exists a vector $\mathbf{y}_k \in Y(\mathbf{x}_{k+1})$ such that

$$\lim_{k\to\infty} \|\varepsilon\mathbf{x}_k+A^T(A\mathbf{z}_{k+1}-\mathbf{b})-\hat{A}^T\mathbf{y}_k\|=0.$$

5. Numerical results. In this section we present the results of some preliminary experiments with the regularization method, the method of Kaczmarz, and the iterative improvement technique. The regularization method refers to the scheme described in the previous section for solving (4.7). However, when solving an unconstrained problem it reduces to the method of Herman et al. [9]. The iterative improvement technique has been obtained from the corresponding regularization method by shifting the origin toward the current solution every ten iterations, leaving the vectors $\mathbf{x}$ and $\mathbf{y}$ in their current positions. The implementation of these methods has been carried out with various values of ε. The starting point in our experiments is always the origin point, i.e., $\mathbf{x}_0=\mathbf{0}$ and $\mathbf{y}_0=\mathbf{0}$. No attempt has been made to find an optimal relaxation parameter, and all the runs have been executed with $w=1$.

The test problems are generated by using a matrix A that has 1000 rows and 3000 columns. Each row of A has only three nonzero elements that have random locations. That is, the column indices of the nonzero elements are random integers between 1 and 3000. The values of the nonzero elements are random numbers from the interval $[-1,1]$. (The random number generators are of uniform distribution.) Let

$$\sigma_i=\mathbf{a}_i^T\mathbf{e}$$

denote the sum of elements in the ith row of A, where $\mathbf{e}=(1,1,\dots,1)^T\in\mathbb{R}^{3000}$. All the test problems have $n=3000$ unknowns, but the number of equations and constraints changes from problem to problem (see the following examples). The definition of the right-hand-side vector ensures that the point $\mathbf{x}^*=\mathbf{e}$ solves the problem. The results of our experiments are given in Tables 1-5, while Table 6 provides the corresponding starting values.

TABLE 1
Solving an inconsistent system of linear equations.

	The value of $(\|A\mathbf{x}_K-\mathbf{b}\|^2-\|A\mathbf{x}^*-\mathbf{b}\|^2)/\|A\mathbf{x}^*-\mathbf{b}\|^2$				
Method	$K=10$	$K=20$	$K=40$	$K=80$	$K=160$
Regularization, $\varepsilon=100$	.379E1	.379E1	.379E1	.379E1	.379E1
Regularization, $\varepsilon=1$	.426E0	.426E0	.426E0	.426E0	.426E0
Regularization, $\varepsilon=10^{-2}$	.818E0	.612E0	.358E0	.132E0	.193E−1
Regularization, $\varepsilon=10^{-4}$	.112E1	.112E1	.111E1	.110E1	.107E1
Regularization, $\varepsilon=10^{-6}$	.113E1	.113E1	.113E1	.113E1	.113E1
Kaczmarz's method	.113E1	.113E1	.113E1	.113E1	.113E1
Iterative improvement, $\varepsilon=1$	.426E0	.856E−1	.126E−1	.180E−2	.299E−3
Iterative improvement, $\varepsilon=0.5$	.174E0	.224E−1	.248E−2	.342E−3	.534E−4
Iterative improvement, $\varepsilon=0.1$	.101E0	.106E−1	.273E−3	.437E−5	.174E−6
Iterative improvement, $\varepsilon=0.05$	.310E0	.945E−1	.949E−2	.191E−3	.126E−6
Iterative improvement, $\varepsilon=0.01$	.818E0	.632E0	.371E0	.138E0	.203E−1

TABLE 2
Solving a constrained least squares problem with inequality constraints.

Method	Values of objective function (upper) and constraints violation (lower) after K iterations.				
	$K=10$	$K=20$	$K=40$	$K=80$	$K=160$
Regularization, $\varepsilon=100$	.795E0	.795E0	.795E0	.795E0	.795E0
	.789E−3	.104E−5	.182E−11	.426E−13	.426E−13
Regularization, $\varepsilon=1$	.213E0	.211E0	.211E0	.211E0	.211E0
	.322E−1	.826E−3	.986E−6	.426E−13	.426E−13
Regularization, $\varepsilon=10^{-2}$	.712E−1	.671E−1	.595E−1	.479E−1	.336E−1
	.796E0	.750E0	.685E0	.558E0	.366E0
Regularization, $\varepsilon=10^{-4}$	.751E−1	.757E−1	.756E−1	.755E−1	.752E−1
	.757E0	.747E0	.747E0	.747E0	.747E0
Regularization, $\varepsilon=10^{-6}$	.751E−1	.757E−1	.756E−1	.755E−1	.752E−1
	.756E0	.747E0	.747E0	.747E0	.747E0
Iterative improvement, $\varepsilon=1$	.213E0	.687E−1	.142E−1	.210E−2	.262E−3
	.322E−1	.197E−1	.264E−3	.201E−4	.218E−5
Iterative improvement, $\varepsilon=0.5$	.115E0	.205E−1	.290E−2	.314E−3	.487E−4
	.130E0	.316E−1	.486E−3	.327E−4	.799E−5
Iterative improvement, $\varepsilon=0.1$	.561E−1	.316E−1	.956E−2	.628E−3	.775E−5
	.560E0	.308E0	.959E−1	.684E−2	.106E−3

TABLE 3
Solving a constrained least squares problem with equality constraints.

Method	Values of objective function (upper) and constraints violation (lower) after K iterations.				
	$K=10$	$K=20$	$K=40$	$K=80$	$K=160$
Regularization, $\varepsilon=100$	.295E−3	.173E−3	.262E−4	.717E−6	.600E−9
	.142E0	.587E−1	.999E−2	.289E−3	.242E−6
Regularization, $\varepsilon=1$	.367E−2	.217E−3	.289E−4	.792E−6	.735E−9
	.160E0	.645E−1	.113E−1	.345E−3	.321E−6
Regularization, $\varepsilon=10^{-2}$	.382E−1	.413E−1	.459E−1	.497E−1	.446E−1
	.106E1	.101E1	.905E0	.725E0	.444E0
Regularization, $\varepsilon=10^{-4}$	.337E−1	.338E−1	.339E−1	.343E−1	.349E−1
	.111E1	.111E1	.110E1	.110E1	.110E1
Regularization, $\varepsilon=10^{-6}$	.336E−1	.336E−1	.336E−1	.336E−1	.336E−1
	.111E1	.111E1	.111E1	.111E1	.111E1
Kaczmarz's method	.336E−1	.336E−1	.336E−1	.336E−1	.336E−1
	.111E1	.111E1	.111E1	.111E1	.111E1
Iterative improvement, $\varepsilon=100$	.295E−3	.195E−3	−.160E−3	.194E−4	.117E−6
	.142E0	.537E−1	.602E−1	.831E−2	.471E−4
Iterative improvement, $\varepsilon=1$	.367E−2	−.118E−2	−.190E−3	.195E−4	.195E−6
	.160E0	.597E−1	.631E−1	.912E−2	.858E−4
Iterative improvement, $\varepsilon=0.1$	.472E−1	.429E−1	.168E−1	.107E−2	.413E−5
	.731E0	.407E0	.207E0	.277E−1	.567E−3

TABLE 4
Solving a constrained least squares problems with mixed constraints.

Method	Values of objective function (upper) and constraints violation (lower) after K iterations.				
	$K=10$	$K=20$	$K=40$	$K=80$	$K=160$
Regularization, $\varepsilon=100$	.124E−1	.123E−1	.121E−1	.121E−1	.121E−1
	.142E0	.587E−1	.999E−2	.289E−3	.242E−6
Regularization, $\varepsilon=1$	.752E−2	.448E−2	.430E−2	.427E−2	.427E−2
	.160E0	.646E−1	.113E−1	.345E−3	.321E−6
Regularization, $\varepsilon=10^{-2}$	.489E−1	.495E−1	.501E−1	.512E−1	.439E−1
	.778E0	.782E0	.819E0	.714E0	.437E0
Regularization, $\varepsilon=10^{-4}$	.479E−1	.483E−1	.485E−1	.486E−1	.488E−1
	.910E0	.909E0	.905E0	.899E0	.886E0
Regularization, $\varepsilon=10^{-6}$	.478E−1	.482E−1	.483E−1	.483E−1	.483E−1
	.912E0	.912E0	.912E0	.912E0	.912E0
Iterative improvement, $\varepsilon=1$	.752E−2	−.182E−3	.150E−3	.561E−4	.226E−6
	.160E0	.467E−1	.631E−1	.931E−2	.992E−4
Iterative improvement, $\varepsilon=0.5$	.197E−1	−.623E−3	−.187E−3	.196E−4	.229E−6
	.201E0	.744E−1	.657E−1	.998E−2	.127E−3
Iterative improvement, $\varepsilon=0.1$	.484E−1	.416E−1	.163E−1	.106E−2	.497E−5
	.731E0	.356E0	.159E0	.276E−1	.490E−3

TABLE 5
Solving a consistent system of linear equations.

Method	The value of $\|Ax_K-\mathbf{b}\|^2$.				
	$K=10$	$K=20$	$K=40$	$K=80$	$K=160$
Regularization, $\varepsilon=100$	.963E3	.963E3	.963E3	.963E3	.963E3
Regularization, $\varepsilon=1$	.200E3	.200E3	.200E3	.200E3	.200E3
Regularization, $\varepsilon=10^{-2}$	.536E0	.282E0	.256E0	.254E0	.254E0
Regularization, $\varepsilon=10^{-4}$	.334E0	.326E−1	.105E−2	.639E−4	.355E−4
Regularization, $\varepsilon=10^{-6}$	.334E0	.325E−1	.982E−3	.263E−4	.822E−6
Kaczmarz's method	.334E0	.325E−1	.982E−3	.263E−4	.817E−6
Iterative improvement, $\varepsilon=0.1$	.113E2	.115E1	.152E0	.228E−1	.221E−2
Iterative improvement, $\varepsilon=0.05$	.400E1	.287E0	.300E−1	.284E−2	.638E−4
Iterative improvement, $\varepsilon=0.01$	.536E0	.625E−1	.742E−2	.896E−4	.134E−6
Iterative improvement, $\varepsilon=0.005$	.487E0	.836E−1	.174E−1	.228E−3	.152E−5
Iterative improvement, $\varepsilon=0.001$	.443E0	.116E0	.238E−1	.445E−3	.103E−4

TABLE 6
Starting values at $\mathbf{x}_0=\mathbf{0}$.

	Example 1	Example 2	Example 3	Example 3	Example 5
Objective function	.400E1	−.140E0	−.100E1	−.100E1	.993E3
Constraints violation		.270E1	.270E1	.270E1	

Example 1: *An inconsistent system of linear equations.* This example considers a system of 1000 equations. The first 500 equations have the form

$$\mathbf{a}_i^T\mathbf{x} = 0.5\sigma_i, \qquad i = 1, \dots, 500,$$

while the other equations are

$$\mathbf{a}_i^T\mathbf{x} = 1.5\sigma_i, \qquad i = 1, \dots, 500.$$

Example 2: *A constrained least squares problem with inequality constraints.* This problem has the form

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{500} (\mathbf{a}_i^T\mathbf{x} - \alpha_i)^2 \\ &\text{subject to} \quad \mathbf{a}_i^T\mathbf{x} \geqq \beta_i \quad \text{for } i = 1, \dots, 500, \end{aligned}$$

where

$$\alpha_i = \min\{0, \sigma_i\},$$

and β_i satisfies the rule: $\beta_i = \sigma_i$ when $\sigma_i \geqq 0$, and $\beta_i = 2\sigma_i$ when $\sigma_i < 0$. The results of this problem are given in Table 2. Here, the objective function value parameter refers to the ratio

$$\left(\sum_{i=1}^{500} (\mathbf{a}_i^T\mathbf{x}_k - \alpha_i)^2 - \sum_{i=1}^{500} (\mathbf{a}_i^T\mathbf{x}^* - \alpha_i)^2\right) \Bigg/ \sum_{i=1}^{500} (\mathbf{a}_i^T\mathbf{x}^* - \alpha_i)^2,$$

while the constraints violation is defined as

$$\max_{1 \leqq i \leqq 500} \{\max\{0, \beta_i - \mathbf{a}_i^T\mathbf{x}_k\}\}.$$

Example 3: *A constrained least squares problem with equality constraints.* In this example, the problem to be solved has the form

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{500} (\mathbf{a}_i^T\mathbf{x} - \alpha_i)^2 \\ &\text{subject to} \quad \mathbf{a}_i^T\mathbf{x} = \sigma_i \quad \text{for } i = 1, \dots 4, 500, \end{aligned}$$

where $\alpha_i = 0$ for $i = 1, \dots, 500$. The objective function value is defined as in the previous example, while the constraints violation parameter is simplified to be

$$\max_{1 \leqq i \leqq 500} |\mathbf{a}_i^T\mathbf{x}_k - \beta_i|.$$

Note that any point that satisfies the constraints is also a solution of the constrained least squares problem. This explains why the regularization method manages to solve this problem when ε is large (see Table 3).

Example 4: *A constrained least squares problem with mixed constraints.* Here we solve the problem

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{500} (\mathbf{a}_i^T\mathbf{x} - \alpha_i)^2 \\ &\text{subject to} \quad \mathbf{a}_i^T\mathbf{x} \geqq \sigma_i \quad \text{for } i \in P \\ &\text{and} \quad \mathbf{a}_i^T\mathbf{x} = \sigma_i \quad \text{for } i \in N, \end{aligned}$$

where

$$\alpha_i = 0, \quad i = 1, \ldots, 500,$$
$$P = \{i \mid 1 \leqq i \leqq 500 \text{ and } \sigma_i \geqq 0\},$$

and

$$N = \{i \mid 1 \leqq i \leqq 500 \text{ and } \sigma_i < 0\}.$$

Here again the objective function value parameter is defined as in Example 2, while the constraints violation parameter is modified to be

$$\max\left\{\max_{i \in P} \{0, \sigma_i - \mathbf{a}_i^T \mathbf{x}_k\}, \max_{i \in N} |\mathbf{a}_i^T \mathbf{x}_k - \sigma_i|\right\}.$$

Example 5: *A consistent system of linear equations.* Although the proposed iterative improvement technique is aimed mainly at solving inconsistent and constrained problems, it is also interesting to watch its behavior when solving a consistent system. Therefore, the last example considers the solution of the system

$$\mathbf{a}_i^T \mathbf{x} = \sigma_i, \qquad i = 1, \ldots, 1000.$$

6. Conclusion. The results of our experiments clearly illustrate that the regularization method has difficulty reaching a least squares solution. Yet, when ε is properly chosen, the iterative improvement process shifts the computed solution toward a least squares solution and overcomes this difficulty.

REFERENCES

[1] A. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[2] Y. Censor, *Row-action methods for huge and sparse systems and their applications*, SIAM Rev., 23 (1981), pp. 444–466.

[3] A. Dax, *The convergence of linear stationary iterative processes for solving singular unstructured systems of linear equations*, SIAM Rev., 32 (1990), pp. 611–635.

[4] L. Elden, *Algorithms for the regularization of ill-conditioned least squares problems*, BIT, 17 (1977), pp. 134–145.

[5] T. Elfving, *An algorithm for maximum entropy image reconstruction from noisy data*, Math. Comput. Modelling, 12 (1989), pp. 729–745.

[6] G. Golub, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[7] R. Gordon, R. Bender, and G. T. Herman, *Algebraic reconstruction techniques* (ART) *for three-dimensional electron microscopy and X-ray photography*, J. Theoret. Biol., 29 (1970), pp. 471–481.

[8] G. T. Herman, A. Lent, and S. W. Rowland, ART: *mathematics and applications*, J. Theoret. Biol., 42 (1973), pp. 1–32.

[9] G. T. Herman, A. Lent, and H. Hurwitz, *A storage-efficient algorithm for finding the regularized solution of a large, inconsistent system of equations*, J. Inst. Math. Appl., 25 (1980), pp. 361–366.

[10] C. Hildreth, *A quadratic programming procedure*, Naval Res. Logist. Quart., 4 (1957), pp. 79–85; Erratum, Ibid, p. 361.

[11] S. Kaczmarz, *Angenaherte Auflosung von Systemen Linearer Gleichungen*, Bull. Acad. Polon. Sci. Lett. A, 35 (1937), pp. 355–357.

[12] A. Lent and Y. Censor, *Extensions of Hildreth's row-action method for quadratic programming*, SIAM J. Control Optim., 18 (1980), pp. 444–454.

[13] J. D. Riley, *Solving systems of linear equations with a positive definite symmetric, but possibly ill-conditioned matrix*, Math. Tables Aids Comput., 9 (1956), pp. 96–101.

[14] K. Tanabe, *Projection method for solving a singular system of linear equations and its applications*, Numer. Math., 17 (1971), pp. 203–214.

[15] A. N. Tikhonov, *The stability of algorithms for the solution of degenerate systems of linear algebraic equations*, J. Comput. Math. Phys., 5 (1965), pp. 181–188.

[16] A. N. Tikhonov and V. Y. Arsenin, *Solutions of ill-posed problems*, V. H. Winston, Washington, 1977.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 585-593, May 1993

PRECONDITIONING THE LANCZOS ALGORITHM FOR SPARSE SYMMETRIC EIGENVALUE PROBLEMS*

RONALD B. MORGAN† AND DAVID S. SCOTT‡

Abstract. A method for computing a few eigenpairs of sparse symmetric matrices is presented and analyzed that combines the power of preconditioning techniques with the efficiency of the Lanczos algorithm. The method is related to Davidson's method and its generalizations, but can be less expensive for matrices that are fairly sparse. A double iteration is used. An effective termination criterion is given for the inner iteration. Quadratic convergence with respect to the outer loop is shown.

Key words. eigenvalues, sparse matrices, Lanczos, preconditioning, Davidson's method

AMS(MOS) subject classifications. 65F15, 15A18

1. Introduction. We consider the problem of computing a few eigenvalues and eigenvectors of a large, sparse, symmetric matrix. It is assumed that factoring the matrix is impractical due to its size and sparsity structure. A method is presented that incorporates both the technique of preconditioning and the Lanczos algorithm. A double iteration scheme is used. The outside loop updates a certain preconditioned matrix; the inside loop applies the Lanczos algorithm. An effective termination criterion is given for the inner loop. This method can be very efficient if the matrix is fairly sparse and an approximate inverse is easily available.

This section briefly discusses background material on eigenvalue techniques and on preconditioning. Section 2 presents the method. Section 3 gives convergence results including asymptotic quadratic convergence with respect to the outer loop. Section 4 gives examples and looks at some implementation details. Comparisons are made with other methods.

Krylov subspace methods are popular for both eigenvalue problems and linear equations problems. Krylov subspaces are used by the Lanczos algorithm [11], [19] for eigenvalues and by the conjugate gradient method [2], [9], [10] for linear equations (see [7, p. 523] and [18] for the relation between the methods). Both methods have convergence problems if the distribution of eigenvalues is unfavorable. The Lanczos algorithm has difficulty if the desired eigenvalues are not well separated from the rest of the spectrum. The conjugate gradient method needs the spectrum to be somewhat separated from zero. Convergence of the conjugate gradient method is often improved by preconditioning (multiplying the matrix equation by an approximate inverse) [1], [3], [8], [13].

It would also be desirable to improve the convergence of the Lanczos algorithm with preconditioning, but this is not straightforward. Preconditioning can be applied indirectly to eigenvalue problems by using the preconditioned conjugate gradient method to solve equations for inverse iteration, the Rayleigh quotient iteration [12], [22], or shift-and-invert Lanczos [6], [20].

Davidson's method [4] and the generated Davidson (GD) method [15], [17] give a more direct approach to preconditioning eigenvalue problems. Suppose eigenvalues are sought for the matrix A. Then these methods generate a subspace with the operator $(M-\rho I)^{-1}(A-\rho I)$, where ρ is the latest approximate eigenvalue obtained by the

* Received by the editors March 23, 1988; accepted for publication (in revised form) April 6, 1992.
† Department of Mathematics, Baylor University, Waco, Texas 76798.
‡ Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, Oregon 97006.

Rayleigh-Ritz procedure [19]. In Davidson's method, the operator M is D, the main diagonal of A. For the GD method, M can be any approximation to A. $M-\rho I$ is in effect a preconditioner for $A-\rho I$. Let (λ, z) be the desired eigenpair. Asymptotically, the subspace produced by these methods resembles a Krylov subspace generated by $(M-\lambda I)^{-1}(A-\lambda I)$. The operator $(M-\lambda I)^{-1}(A-\lambda I)$ has one eigenvalue at 0 with z as the associated eigenvector. The rest of the spectrum tends to be compressed around 1 by the preconditioning. This makes 0 well separated, and convergence is rapid toward z (see [17] for more detail).

The GD method is more expensive per iteration than the Lanczos algorithm. There is the cost of the preconditioning and also the cost of the Rayleigh-Ritz procedure. Full orthogonalization must be done. Davidson's method is often applied to problems for which the main expense is the matrix-vector products, and thus any reduction in the number of iterations is worthwhile. For other problems, particularly those where A is quite sparse, the orthogonalization costs are significant. Restarting reduces the expense but slows down the convergence.

A method is desired that could be preconditioned and could take advantage of the Lanczos recurrence. One possibility involves using the Lanczos algorithm to build the subspace for GD [16, pp. 71-83]. The Rayleigh-Ritz expense can be reduced by a factor of five. However, a double iteration is used and the cost still grows as the size of the subspace increases. In the next section, another method is presented that uses preconditioning and the Lanczos algorithm. The difference is that in this method the Rayleigh-Ritz procedure is not done with respect to A. A double iteration is again used, but the costs of order n are fixed as the subspace grows.

2. The Preconditioned Lanczos (PL) Algorithm.

Preconditioning the Lanczos algorithm was suggested by Scott [21] as a special case of a method for generalized eigenvalue problems, but it was not investigated, and the algorithm has not been given before for the standard eigenvalue problem. Our purpose here is threefold. We establish important convergence results. Scott's results [21] do not take into account preconditioning or early termination of the inner iteration. Second, we implement the method and show that it is useful. Third, we derive the method showing its connection with the Davidson and GD methods. This is important because it gives insight into why the method is effective and because Davidson's method is well known among quantum chemists.

The GD method uses the operator $N(\rho)\equiv(M-\rho I)^{-1}(A-\rho I)$ to generate a subspace, but it uses the operator A in the Rayleigh-Ritz procedure for extracting approximate eigenvectors from the subspace. However, since $N(\rho)$ has an eigenvector approximating one of A, the Rayleigh-Ritz procedure with a fixed $N(\rho)$ is also useful for computing an eigenvector of A. It is necessary to restart the Rayleigh-Ritz occasionally with a new ρ, because the eigenvector of $N(\rho)$ is only an approximation. We use this idea, but transform $N(\rho)$ to a symmetric operator so that the Lanczos algorithm can be applied. A positive definite preconditioner is required, so we replace $M-\rho I$ with M_k. The algorithm is called the PL Algorithm.

THE PL ALGORITHM. Given a vector x_0, compute $\rho_0 = x_0^T A x_0 / x_0^T x_0$, and FOR $k=0,1,2,\dots$, DO 1 to 5

1. Choose M_k to be a positive definite matrix approximating $A-\rho_k I$, and let $L_k L_k^T$ be its Cholesky factorization.
2. Define $W_k \equiv L_k^{-1}(A-\rho_k I)L_k^{-T}$.

3. Run the Lanczos iteration with W_k and starting vector $L_k^T x_k$, until the smallest Ritz value is bounded away from zero by the residual norm [19, p. 260]. Letting θ_k be the smallest Ritz value and y_k be the corresponding Ritz vector of unit length, this stopping test is $-\theta_k > \| W_k y_k - \theta_k y_k \|$.
4. Let $x_{k+1} = L_k^{-T} y_k$.
5. Set $\rho_{k+1} = x_{k+1}^T A x_{k+1} / x_{k+1}^T x_{k+1} = \rho_k + \theta_k / x_{k+1}^T x_{k+1}$.

The operator $W_k \equiv L_k^{-1}(A - \rho_k I) L_k^{-T}$ is related to $M_k^{-1}(A - \rho_k I)$ by a similarity transformation. And $M_k^{-1}(A - \rho_k I)$ has an eigenvector approximating an eigenvector of A. By applying the Lanczos iteration to W_k, computing a Ritz vector, and then transforming it back, we get x_{k+1}, an approximate eigenvector of A. The Rayleigh quotient ρ_{k+1} of x_{k+1} is an approximate eigenvalue. The residual norm [19, p. 69] of (ρ_{k+1}, x_{k+1}) with respect to A can be monitored to determine a stopping point for the PL iteration.

The main cost of the method is in the Lanczos loop. A matrix-vector product with A is required and systems of linear equations in L_k and L_k^T are solved. So it is important for the Lanczos iteration to converge quickly. The spectrum of W_k, or equivalently of $M_k^{-1}(A - \rho_k I)$, is the determining factor. Here the preconditioning improves the eigenvalue distribution just as it does for the GD method. The early termination test in step 3 of the PL Algorithm also reduces the number of Lanczos iterations. In the following section it is shown that the early termination test does not significantly impair the convergence of the outer iteration.

3. Convergence of PL. For convergence, the preconditioner does not need to be an accurate approximation. If there is boundedness, then the sequence $\{\rho_k\}$ in the PL Algorithm converges to an eigenvalue of A at an asymptotically quadratic rate. First, convergence of ρ_k to an eigenvalue is shown.

THEOREM 1. *Assume that both M_k and M_k^{-1} are uniformly bounded in norm. Then ρ_k converges to an eigenvalue of A.*

Proof. First we will establish the equality asserted in step 5 of the PL Algorithm.

$$
\begin{aligned}
\rho_{k+1} &= x_{k+1}^T A x_{k+1} / x_{k+1}^T x_{k+1} \\
&= \rho_k + (L_k^{-T} y_k)^T (A - \rho_k I) L_k^{-T} y_k / x_{k+1}^T x_{k+1} \\
&= \rho_k + y_k^T W_k y_k / x_{k+1}^T x_{k+1} \\
&= \rho_k + \theta_k / x_{k+1}^T x_{k+1}.
\end{aligned} \tag{1}
$$

Next we will show that θ_k is nonpositive, so that from (1), $\{\rho_k\}$ is a nonincreasing sequence. The $(1, 1)$ element of the tridiagonal matrix T that is generated by the Lanczos iteration is the Rayleigh quotient of the starting vector $L_k^T x_k$ with respect to W_k. It can be seen that this Rayleigh quotient is zero, since $\rho_k = x_k^T A x_k / x_k^T x_k$. Using Cauchy's interlace theorem, θ_k is less than or equal to zero, since θ_k is the smallest eigenvalue of T.

The sequence of ρ_k's is bounded below by the smallest eigenvalue of A. Therefore, ρ_k converges. Let the limit be τ, and let $\varepsilon_k = \rho_k - \tau$.

Because by assumption M_k^{-1} is bounded in norm, so also is L_k^{-1}. The vector y_k is of unit length, so $x_{k+1}^T x_{k+1} = (L_k^{-T} y_k)^T L_k^{-T} y_k$ is bounded. From (1),

$$
|\theta_k| = (\rho_k - \rho_{k+1}) x_{k+1}^T x_{k+1} \leqq \varepsilon_k x_{k+1}^T x_{k+1}. \tag{2}
$$

Therefore, θ_k converges to zero. But θ_k is within the residual norm bound of some eigenvalue of W_k, say ω_k. Because of the stopping test for the Lanczos iteration in step 3 of the PL algorithm, ω_k is within $|\theta_k|$ of θ_k and so within $2|\theta_k|$ of zero. Therefore, ω_k converges to zero.

Since W_k has an eigenvalue converging to zero, $\|W_k^{-1}\|$ goes to infinity. Since

$$\|W_k^{-1}\| = \|L_k^T(A-\rho_k I)^{-1}L_k\| \leqq \|L_k^T\|\,\|(A-\rho_k I)^{-1}\|\,\|L_k\|,$$

and $\|L_k^T\|$ and $\|L_k\|$ are bounded, ρ_k must be converging to an eigenvalue of A, and the proof is complete. □

The theorem does not say which eigenvalue ρ_k converges to, but it is extremely likely to be the smallest one. For this not to be the case would require that each starting vector of the Lanczos iteration have a very small component in the direction of the eigenvector associated with the smallest eigenvalue of W_k.

Let ρ_k converge to λ. The next theorem shows that this convergence is asymptotically quadratic. The "big O" notation will be used: $\beta = \gamma + O(\varepsilon)$ implies $|\beta - \gamma| \leqq c\varepsilon$ for $c > 0$.

THEOREM 2. *Assume that both M_k and M_k^{-1} are uniformly bounded in norm. Then ρ_k converges at an asymptotically quadratic rate.*

Proof. Using the definitions of W_k and y_k,

$$\begin{aligned}(A-\rho_k I)x_{k+1} &= L_k W_k L_k^T x_{k+1} = L_k W_k y_k \\ &= L_k[(W_k y_k - \theta_k y_k) + \theta_k y_k].\end{aligned}$$

Because of the stopping test in step 3 of the PL Algorithm and the fact that y_k is a unit vector,

$$\begin{aligned}\|(A-\rho_k I)x_{k+1}\| &\leqq \|L_k\|[\|W_k y_k - \theta_k y_k\| + |\theta_k|\,\|y_k\|] \\ &\leqq 2\|L_k\|\,|\theta_k|.\end{aligned} \tag{3}$$

Let the residual vector of x_{k+1} with respect to A be r_{k+1}:

$$\|r_{k+1}\| = \|(A-\rho_{k+1}I)x_{k+1}\| / \|x_{k+1}\|.$$

Using first the minimal residual property [19, p. 12], then (3),

$$\begin{aligned}\|r_{k+1}\| &\leqq \|(A-\rho_k I)x_{k+1}\| / \|x_{k+1}\| \\ &\leqq 2\|L_k\|\,|\theta_k| / \|x_{k+1}\|.\end{aligned}$$

With (2) and the fact that $\|L_k\|$ is bounded and $\|x_{k+1}\|$ is bounded away from zero, we have

$$\|r_{k+1}\| = O(\varepsilon_k). \tag{4}$$

Let γ_k be the gap between ρ_k and the nearest eigenvalue of A other than λ. This gap approaches a nonzero constant because ρ_k converges to λ. By the Kato-Temple inequality [19, p. 222],

$$|\rho_{k+1} - \lambda| \leqq \|r_{k+1}\|^2 / \gamma_{k+1}.$$

With (4), this becomes

$$\rho_{k+1} = \lambda + O(\varepsilon_k^2),$$

and we have quadratic convergence. □

4. Implementation and examples. Here examples are given and comparisons are made with other methods. First, some implementation details are discussed.

On the choice of preconditioner, one possibility is to pick M to be a portion of A. Then factor $M-\rho_k I$ in LDL^T form, and let M_k equal $L|D|L^T$. Small pivot elements during the factorization should be changed. Another approach is to let M_k be $M-\sigma I$, where σ is an estimate of the smallest eigenvalue of A and is below the spectrum of M. A powerful preconditioning approach is to use incomplete factorization [8], [13] of $A-\sigma I$. Here it is important that $A-\sigma I$ be positive definite. Even then, adjustment of small or negative pivot elements may be necessary for a stable factorization. The symmetric successive overrelaxation (SSOR) [9] preconditioner is another possibility.

If more than one eigenvalue is desired, a form of deflation can be used. The eigenvalues that are already computed can be shifted out of the way. For example, once (λ_1, z_1) has been computed, $A-\rho_k I$ can be replaced by $A-\rho_k I+\gamma z_1 z_1^T$, where γ is large enough to shift λ_1 away from the other desired eigenvalues. The shifting should be kept as small as possible to comfortably achieve this, because a large shift and the effect of M_k^{-1} could possibly produce a large eigenvalue for W_k. This would slow the convergence of the Lanczos loop. The starting vector for finding the second eigenvalue can be determined while the first eigenvalue is being computed. At some point, the second Ritz vector of W_k can be computed and multiplied by L_k^{-T}. In the examples that follow, the starting vectors for interior eigenpairs are calculated once the residual norm for the previous eigenpair reached two-thirds in orders of magnitude of the desired accuracy.

An additional stopping test is applied in the Lanczos loop to terminate early if it is likely that the desired accuracy has been attained. This is done by comparing the improvement in the residual norm with respect to W_k of the Ritz vector in the Lanczos loop with the improvement needed in the residual norm with respect to A of the approximate eigenvector. (The Lanczos loop is terminated when the log base 10 of the ratio of the residual norms of Ritz vectors at the beginning and current point of Lanczos is less than the log of the ratio of the residual norm of x_k to the specified residual tolerance, divided by a safety factor of 0.9 times the ratio of logs of improvements in residual norms for the previous k.) In our testing, this check for early termination helps the method avoid extra computation.

Example 1. The first test matrix is $A=\mathrm{Diag}(1,2,3,\dots,1000)$. For the preconditioner, let $M=\mathrm{Diag}(10.1, 10.2, 10.3,\dots,110)$ and $M_k=M-\rho_k I$. The starting vector is $(1, 1/2, 1/3,\dots,1/1000)^T$. The smallest eigenvalue and eigenvector are computed. The requested residual tolerance is 1×10^{-8}. Table 1 gives the results of the PL method. It lists the number of iterations in the Lanczos loop (this is the size of the Krylov subspace generated and also the number of matrix-vector products with A), the new approximate eigenvalue ρ_{k+1}, and the residual norm. It shows the expected quadratic convergence. The total number of iterations is 88. The standard Lanczos algorithm requires 194 iterations for the same task. So the preconditioning cuts the number of

TABLE 1
PL *for Example* 1.

k	Iterations	ρ_{k+1}	Residual norm
		4.55	24.2
0	3	1.67	4.5
1	8	1.064	1.0
2	13	1.00035	0.101
3	25	1.0000000058	0.56E−3
4	39	1.0000000000	0.76E−8

iterations in half. With better preconditioning, this can be improved. For example, with $M = \text{Diag}(1.1, 1.2, 1.3, \ldots, 101)$, the PL Algorithm converges in only 30 total iterations.

Next we compare three methods that use preconditioning: the PL Algorithm, the GD method, and the Rayleigh quotient iteration (RQI) [12], [22] with preconditioned SYMMLQ [18] in the inner loop. The GD method has a limit of 40 on the size of the subspace because of the orthogonalization costs. The RQI uses termination criterion in SYMMLQ of $\text{RTOL} = \text{Min}\{0.01 * \|r\|, \|r\|^3\}$, where $\|r\|$ is the residual norm at the previous step of RQI. The choice of termination criterion can affect the convergence a great deal, especially when the starting vector for RQI is not accurate. Three matrices with increasing difficulty are used. Let $A = \text{Diag}(1, 1+\delta, 1+2\delta, \ldots, 1+99\delta, 2+99\delta, 3+99\delta, \ldots, 901+99\delta)$, for $\delta = 1$, 0.1, and 0.01. M is as originally defined. Table 2 gives the total number of matrix-vector products used for each of these methods. For the toughest problem ($\delta = 0.01$), the PL Algorithm is much better than the GD method. Not only are fewer iterations required, but each iteration is considerably less expensive. The PL Algorithm has the advantage of being able to cheaply generate large subspaces, and for difficult problems, large subspaces are needed. The PL Algorithm also has an advantage when storage is limited and the vectors spanning the subspace must be stored on disk, because the GD method accesses these vectors at every iteration.

TABLE 2
Preconditioning methods.

	Number of matrix-vector products		
	Matrix	Matrix	Matrix
Method	$\delta = 1$	$\delta = 0.1$	$\delta = 0.01$
PL	88	247	555
GD	69	309	1584
RQI	99	333	582

The PL Algorithm and RQI are fairly comparable in this example. This is not surprising, since the two methods do resemble each other. The main difference is that the PL Algorithm has an eigenvalue problem in the inner loop instead of a linear equations problem. RQI does not require storage of the vectors spanning the subspace and has easier deflation. But the PL Algorithm has the advantage of being more reliable in converging to the smallest eigenvalue. To illustrate this with the first matrix ($\delta = 1.0$) and starting vector $(5, 5, 5, 5, 5, 1/6, 1/7, 1/8, \ldots, 1/1000)^T$, RQI converges to the fifth eigenvalue. The PL Algorithm still converges to the first eigenvalue. Of course, it is well known that to insure convergence to a particular eigenvalue, RQI needs a good starting vector or an implementation that combines it with inverse iteration.

The shift-and-invert Lanczos method with the preconditioned conjugate gradient method can also be used, but we have not found it to be competitive. For one implementation with early termination of the inner conjugate gradient iteration, 893 iterations were required with the matrix $\delta = 0.1$. However, there is probably a better implementation.

Example 2. This and the next example use matrices from the Harwell-Boeing sparse matrix collection [5]. The matrices selected are quite sparse and have small eigenvalues that are not well separated. The first matrix is SHERMAN1, from oil reservoir simulation. The dimension is 1000, and it has seven nonzero bands with half-bandwidth of 100. The six smallest eigenvalues are 0.000323, 0.00102, 0.00111, 0.00151, 0.00192, and 0.00205, and the largest is 5.04.

The elements of the starting vector are selected randomly on the interval $(0, 1)$. The computations are performed on an IBM 3090-170J computer using double precision arithmetic. The Lanczos algorithm is restarted every 250 steps and the GD method every 40. The small eigenvalue problems generated by these methods are solved with EISPACK routines, though an iterative method such as subspace iteration [19] might be better. The small problem is solved at the end of each 250 iterations of Lanczos, at every step for GD (as is required), and for PL at each of the first seven iterations and then at multiples of five. A few steps of inverse iteration are needed at the start of RQI. We let $\rho = 0$ for the first three outer iterations, then switch to regular RQI. The preconditioner for all methods is from incomplete factorization [13] of A, with no fill-in allowed.

First, the smallest eigenpair of the matrix is computed with residual norm tolerance of 1×10^{-8}. Table 3 lists the total number of matrix-vector products (MVPs) with A, along with the time in CPU seconds. The preconditioning methods need far fewer iterations than the Lanczos algorithm. The PL Algorithm is the fastest method, even though the GD method requires fewer iterations.

TABLE 3
Harwell-Boeing matrices.

	SHERMAN1		SHERMAN1		NOS6	
	One eigenpair		Five eigenpairs		One eigenpair	
Method	MVPs	Time	MVPs	Time	MVPs	Time
Lanczos	750	7.6	—		—	
PL	61	1.2	487	9.3	987	16.7
GD	39	2.2	149	9.7	6920	346
RQI	221	4.2	—		—	

Next the five smallest eigenpairs are computed. PL uses $\gamma = 0.01$ to shift computed eigenvalues. The results are not very sensitive to the choice of γ. (For example, with $\gamma = 0.002$, 445 matrix-vector products are needed, while PL with $\gamma = 0.1$ uses 533.) The PL and GD methods require about the same amount of time. The simple Lanczos algorithm with restarting has difficulty calculating these eigenvalues. A deflation scheme is possible, but it might be better to use a block Lanczos method [6] or out-of-core storage to avoid restarting. RQI is implemented with $\rho = 0$ for the first three outer iterations for each eigenvalue and with a small Raleigh-Ritz procedure [19] applied to the last five approximate eigenvectors to get each new starting vector. RQI finds the first, third, fifth, second, and ninth eigenvalues and uses 2001 total matrix-vector products. A different implementation might help, but this appears to be a difficult problem for RQI.

Example 3. Here we use a structural engineering matrix of dimension 675 called NOS6 in Simon's LANPRO collection [5]. The starting vector has entries distributed randomly on the interval $(-1, 1)$, and the residual tolerance is 1×10^{-4}. The two smallest eigenvalues are 1.0000153 and 1.0000254, which are extremely close considering that the matrix has elements of size 4×10^{6}. The maximum size subspace for the Lanczos algorithm and for the Lanczos iteration of PL is 350, while GD again uses dimension 40. The results for computing one eigenvalue are given in Table 3. The PL Algorithm is more than 20 times faster than GD. The Lanczos algorithm shows little sign of ever finding the smallest eigenvalue. RQI is implemented with initial steps of inverse iteration using $\rho = 0$ until the Rayleigh quotient is less than 1.00002, but it also cannot find this eigenvalue (it never switches from inverse iteration to RQI). In another test with $\rho = 1$ initially, RQI converges in 1961 total iterations and 30.2 seconds. So even with an extremely accurate initial estimate of the eigenvalue, RQI is slower than the PL Algorithm.

5. Conclusion.

Like the Davidson and GD methods, the PL Algorithm uses preconditioning to accelerate the convergence. However, the expense per iteration is reduced by applying the Lanczos iteration. A double iteration scheme is used. The inner iteration is terminated early for efficiency, but convergence is still asymptotically quadratic with respect to the outer loop.

The PL Algorithm can significantly reduce the expense of computing eigenvalues and eigenvectors of some matrices. It is most useful for sparse matrices, for difficult problems, and for computing only a very small number of extreme eigenpairs.

We conclude with some possibilities for further research. If an eigenvalue estimate σ is known, $A - \rho_k I$ can be replaced with $A - \sigma I$ in step 2 of the PL Algorithm, for the first few outer loops. Also, the method might be more robust if the x_k vectors are saved and combined using the Rayleigh-Ritz procedure with A. This would not be too expensive if only the last few x_k vectors are used. Finally, an interior implementation [14] of the Lanczos loop might allow computation of more than one eigenvalue without using deflation. This also might allow computation of several eigenvalues at a time if estimates are known of the desired eigenvalues.

Acknowledgments. The authors would like to thank the referees for many suggestions that considerably improved the paper.

REFERENCES

[1] O. B. Axelsson, *On the efficiency of a class of A-stable methods*, BIT, 14 (1974), pp. 279-287.

[2] A. Cline, *Several observations on the use of conjugate gradient methods*, ICASE Report 76-22, NASA Langley Research Center, Hampton, VA, 1976.

[3] 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.

[4] E. R. Davidson, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices*, J. Comput. Phys., 17 (1975), pp. 87-94.

[5] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1-14.

[6] T. Ericsson and A. Ruhe, *The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems*, Math. Comp., 35 (1980), pp. 1251-1268.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.

[8] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142-156.

[9] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[10] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409-436.

[11] C. Lanczos, *An iterative method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45 (1950), pp. 255-282.

[12] J. G. Lewis, *Algorithms for sparse matrix eigenvalue problems*, Ph.D. thesis, Stanford University, Stanford, CA, 1977.

[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] R. B. Morgan, *Computing interior eigenvalues of large matrices*, Linear Algebra Appl., 154-156 (1991), pp. 289-309.

[15] R. B. MORGAN, *Davidson's method and preconditioning for generalized eigenvalue problems*, J. Comput. Phys., 89 (1990), pp. 241-245.
[16] ———, *Preconditioning eigenvalue problems*, Ph.D. thesis, University of Texas, Austin, TX, 1986.
[17] R. B. MORGAN AND D. S. SCOTT, *Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 817-825.
[18] C. C. PAIGE AND M. A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617-629.
[19] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
[20] D. S. SCOTT, *The advantages of inverted operators in Rayleigh-Ritz approximations*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 68-75.
[21] ———, *Solving sparse symmetric generalized eigenvalue problems without factorization*, SIAM J. Numer. Anal., 18 (1981), pp. 102-110.
[22] D. B. SZYLD, *Criteria for combining inverse and Rayleigh quotient iteration*, SIAM J. Numer. Anal., 25 (1988), pp. 1369-1375.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 594-606, May 1993

THE HERMITE SPECTRAL METHOD FOR GAUSSIAN-TYPE FUNCTIONS*

TAO TANG†

Abstract. Although Hermite functions were widely used for many practical problems, numerical experiments with the standard (normalized) Hermite functions $\psi_n(v)$ worked poorly in the sense that too many Hermite functions are required to solve differential equations. In order to obtain accurate numerical solutions, it is necessary to choose a scaling factor α and use $\psi_n(\alpha v)$ as the basis functions. In this paper the scaling factors are given for functions that are of Gaussian type, which have finite supports $[-M, M]$. The scaling factor used is $\max_{0\leq j\leq N}\{\gamma_j\}/M$, where $\{\gamma_j\}_{j=0}^N$ are the roots of $\psi_{N+1}(v)$ and $N+1$ is the number of the truncated terms used. The numerical results show that after using this scaling factor, only reasonable numbers of the Hermite functions are required to solve differential equations.

Key words. spectral method, Hermite functions

AMS(MOS) subject classifications. 65M70, 33C25, 35K05

1. Introduction.

The normalized Hermite functions are

$$\psi_n(v)=\frac{1}{\sqrt{2^n n!}}H_n(v)\exp\left(-\frac{v^2}{2}\right), \tag{1.1}$$

where the $H_n(v)$ are the usual (unnormalized) Hermite polynomials. For physical problems posed on $v\in(-\infty,\infty)$ (i.e., infinite domain), a variety of spectral techniques have been developed in recent years. These include the use of Fourier series combined with domain truncation, sine function, Hermite functions, and algebraically mapped Chebyshev polynomials. Previous results for these techniques are summarized, for example, in Boyd [5, Chap. 14]. Many researchers have noticed that the close connection of Hermite functions to physics makes them a natural choice of basis functions for many fields of science and engineering. Numerical applications include many problems in continuum mechanics (see, e.g., [3]), particle physics (see, e.g., [9], [15], and [17]), tropical meteorology, and oceanography (see, e.g., [2], [8], and [14]). Also, one reason for using Hermite spectral methods is that the Hermite system has some very attractive properties from the numerical point of view. For example, in a recent paper, Weideman [19] showed that the spectral radii for the first and second Hermite differentiation matrices are $O(\sqrt{N})$ and $O(N)$, respectively, where $N+1$ is the number of truncated terms used. This places rather weak stability restrictions on the Hermite method. For example, if we consider the standard heat equation, then a maximum step size in the time direction of order $O(N^{-1})$ is required, whereas for Fourier and Chebyshev methods it is of order $O(N^{-2})$ and $O(N^{-4})$, respectively. In the actual calculations this means that we need not even consider implicit time integration methods with the Hermite method. A theoretical study of the Hermite method for the heat equation is given in a recent paper, [11], which is concerned with the stability and convergence properties of the method.

Although the Hermite spectral methods have some attractive properties, the direct spectral approach may not produce good approximations. In the practical calculations,

* Received by the editors November 25, 1991; accepted for publication (in revised form) April 1, 1992.

† Department of Mathematics and Statistics, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6. This work was supported by Natural Sciences and Engineering Research Council of Canada grant OGP0105545.

it is necessary to choose a scaling factor α since we can always use $\psi_n(\alpha v)$ as the basis set for any finite α. This freedom does not exist for a finite interval, but after a change of scale in v, an infinite domain is still an infinite domain. It was pointed out by Boyd [4]-[6] that α should increase with the truncated terms used, but the theory of choosing an optimum scaling factor is still incomplete. In this paper, we study how to choose a proper scaling factor for a class of functions that decay at infinity at least like $\exp(-pv^2)$ for some positive constants p (i.e., Gaussian type). Solutions of many practical problems behave like Gaussian-type functions at infinity, for example, the diffusion equations for heat flow and the Fokker-Planck equations for particle physics (see, e.g., [10] and [17]). The idea underlying our approach is the following: In using spectral collocation methods we choose a scaling factor α which depends on the ratio of the maximum root of $H_{N+1}(v)$ and the length of the finite support of the function. The usual method for dealing with the infinite interval is simply to truncate $(-\infty, \infty)$ to a finite interval $[-M, M]$. The scaling factor α is chosen so that all of the collocation points are within $[-M, M]$. The numerical results show that after using the scaling factor, reasonable numbers of Hermite functions are required to resolve a Gaussian-type function.

2. Hermite collocation methods. For a Gaussian-type function f, we have the expansion

$$f(v) = \sum_{n=0}^{\infty} a_n \psi_n(\alpha v), \qquad |v| < \infty, \tag{2.1}$$

which is equivalent to

$$f(v/\alpha) = \sum_{n=0}^{\infty} a_n \psi_n(v), \qquad |v| < \infty, \tag{2.2}$$

where α is a positive constant. The spectral method of order N is to approximate the function f using the first $N+1$ terms in the expansion series, i.e., the coefficients $\{a_n\}_{n=N+1}^{\infty}$ are set equal to 0. Therefore, we have the spectral approximations

$$f_N(v/\alpha) = \sum_{n=0}^{N} a_n \psi_n(v). \tag{2.3}$$

When solving differential equations, we need to relate the coefficients of the derived function to those of the original function, as given by

$$f^{(k)}(v/\alpha) = \sum_{n=0}^{N} a_n \psi_n^{(k)}(v) = \sum_{n=0}^{N} a_n^{(k)} \psi_n(v). \tag{2.4}$$

Using the recurrence formulas of Hermite functions we can obtain that

$$a_n^{(1)} = \frac{\alpha}{\sqrt{2}} \left(-\sqrt{n}\, a_{n-1} + \sqrt{n+1}\, a_{n+1}\right), \tag{2.5}$$

$$a_n^{(2)} = \frac{\alpha^2}{2} \left(\sqrt{(n-1)n}\, a_{n-2} - (2n+1) a_n + \sqrt{(n+1)(n+2)}\, a_{n+2}\right), \tag{2.6}$$

with $a_n = 0$ whenever $n < 0$ or $n > N$.

In pseudospectral methods, the optimum pseudospectral points are the roots of $H_{N+1}(v)$, which are denoted by $\{\gamma_j\}_{j=0}^{N}$ with the order $\gamma_0 > \gamma_1 > \cdots > \gamma_N$. Assuming that (2.3) is satisfied at the collocation points, we have

$$f_N\left(\frac{\gamma_j}{\alpha}\right) = \sum_{n=0}^{N} a_n \psi_n(\gamma_j), \qquad 0 \leq j \leq N. \tag{2.7}$$

Noting that

$$\sum_{n=0}^{N} \psi_n(\gamma_i)\psi_n(\gamma_j) = C_i\delta_{ij}, \qquad 0 \leqq i, j \leqq N, \tag{2.8}$$

$$C_i = \sum_{n=0}^{N} [\psi_n(\gamma_i)]^2, \qquad 0 \leqq i \leqq N, \tag{2.9}$$

we obtain from (2.7) that

$$a_n = \sum_{j=0}^{N} \frac{1}{C_j} f_N\left(\frac{\gamma_j}{\alpha}\right)\psi_n(\gamma_j), \qquad 0 \leqq n \leqq N. \tag{2.10}$$

The above relations between $\{a_n\}$ and $\{f_N(\gamma_j/\alpha)\}$ give a simple evaluation of the derivatives of $f_N(v)$ at the points $\{\gamma_j/\alpha\}_{j=0}^N$. Once $\{f_N(\gamma_j/\alpha)\}_{j=0}^N$ are known, the coefficients $\{a_n\}_{n=0}^N$ can be obtained by (2.10). Then, the first and second derivatives of f_N at points $\{\gamma_j/\alpha\}_{j=0}^N$ can be computed by (2.4)-(2.6). A higher order of derivatives can be obtained in a similar way. If $\{a_n\}_{n=0}^N$ are given, the function $f_N(v)$ is computed by

$$f_N(v) = \sum_{n=0}^{N} a_n\psi_n(\alpha v). \tag{2.11}$$

3. The scaling factor. Suppose that the function f has a finite support $[-M, M]$, i.e., $f(v) \sim 0$ for $|v| > M$. In order to compute $\{a_n\}_{n=0}^N$ by (2.10), we need to use information from the interval $[-M, M]$ only, since outside this region the function is almost zero and will not contribute much to a_n. This simple motivation suggests that

$$\left|\frac{\gamma_j}{\alpha}\right| \leqq M, \quad \text{for all } 0 \leqq j \leqq N. \tag{3.1}$$

The above condition is satisfied by choosing

$$\alpha = \alpha_N = \max_{0 \leqq j \leqq N} \{\gamma_j\}/M = \gamma_0/M. \tag{3.2}$$

Since $\gamma_0 \sim \sqrt{2N}$ (see, e.g., [1]), we obtain that

$$\alpha_N \sim \sqrt{2N}/M. \tag{3.3}$$

The Hermite spectral methods were rejected before because of their poor resolution properties. Gottlieb and Orszag [12] investigated the rate of convergence of Hermite series by considering the expansion of the sine functions. They found that to resolve p wavelengths of the sine function requires $O(p^2)$ polynomials. In the case when the function decays rapidly, we need to consider the sine functions which oscillate rapidly in a finite interval (i.e., $|v| \leqq \text{Const.}$). Consider

$$\sin(pv) = \sum_{n=0}^{\infty} c_{2n+1}H_{2n+1}(v), \tag{3.4}$$

where

$$c_{2n+1} = (-1)^n \frac{p^{2n+1}\exp(-p^2/4)}{2^{2n+1}(2n+1)!}, \tag{3.5}$$

for $n = 0, 1, \ldots$. Using the asymptotic expansion of $H_n(v)$ [1],

$$H_n(v) \sim \exp(v^2/2)\frac{n!}{(\frac{1}{2}n)!}\cos\left(\sqrt{2n+1}\, v - \frac{1}{2}n\pi\right), \tag{3.6}$$

and Sterling's formula

$$n! \sim \sqrt{2\pi}\, n^{n+1/2} \exp(-n), \tag{3.7}$$

we obtain the asymptotic behavior of the nth term of the right-hand side of (3.4) as

$$\text{term}_n \sim \frac{1}{\sqrt{2\pi}} \left(\frac{p^2}{4n}\right)^{n+1/2} \exp(n - p^2/4 + v^2/2). \tag{3.8}$$

It can be seen from the above equation that if $n \approx p^2/4$ then the nth term behaves like $\exp(v^2/2)/\sqrt{2\pi}$, which is in general not small. That is, to resolve p wavelengths of the sine functions requires more than $0.25p^2$ Hermite functions. This result is similar to that observed by Gottlieb and Orszag [12]. However, if we use the scaling factor α_N of the form (3.2), i.e., if we expand

$$\sin(pv) = \sum_{n=0}^{\infty} d_{2n+1} H_{2n+1}(\alpha_N v) \tag{3.9}$$

with $\alpha_N = \gamma_0$, then the nth expansion term is of the asymptotic form

$$\text{term}_n \sim \frac{1}{\sqrt{2\pi}} \left(\frac{p^2}{4n\alpha_N^2}\right)^{n+\frac{1}{2}} \exp\left(n - \frac{p^2}{4\alpha_N^2} + \frac{v^2}{2}\right). \tag{3.10}$$

Noting that $\alpha_N^2 \sim 2N$, we obtain

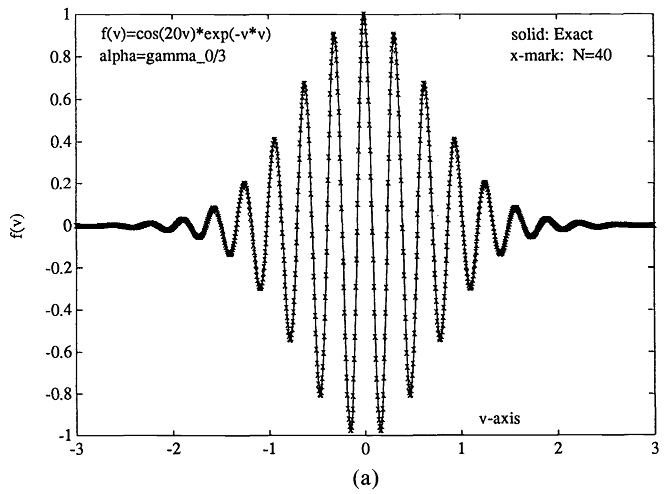
$$\text{term}_n \sim \frac{p}{4\sqrt{\pi n N}} \left[\frac{p^2}{8nN} \exp\left(1 - \frac{p^2}{8nN}\right)\right]^n \exp\left(\frac{v^2}{2}\right). \tag{3.11}$$

The right-hand side of the above equation decays exponentially when $n \geqq N \approx p$. This result shows that using the scaling factor can reduce the number of required polynomials to the best possible case. The above analysis is only given for the expansion in Hermite polynomials, but similar behavior can be seen for the expansion in the normalized Hermite functions (in this case, if the function has a finite support $[-M, M]$, then $N \approx Mp$). To see this, we expand

$$\cos(pv)\exp(-v^2) = \sum_{n=0}^{\infty} a_n \psi_n(\alpha v). \tag{3.12}$$

The coefficients $\{a_n\}_{n=0}^N$ are obtained by using (2.10) and the numerical curve is given by (2.11). It can be seen from Fig. 1 that, after scaling (i.e., $\alpha = \alpha_N = \gamma_0/3$), 20 and 30 expansion terms give good approximations to the function $\cos(pv)\exp(-v^2)$ with $p = 20$ and $p = 30$ (noting that the function $\cos(pv)\exp(-v^2)$ is an even function and the number of the expansion term is $N/2$ since the odd terms are zero). However, approximations without scaling (i.e., $\alpha = 1$) require more than 100 terms in the case $p = 20$; see Fig. 2. Furthermore, in Figs. 3 and 4, we have plotted the absolute values of the even coefficients in (3.12) (noting that the odd coefficients are zero) as a function of n in the case $p = 20$. In these two figures a log scale was used. Since $\psi_n(\alpha v) = O(n^{-1/4})$, the nth term of (3.12) is small only when its coefficient is small. Figure 3 suggests that in order to have the nth term less than 10^{-3}, for all $n \geqq N$, we need about 25 expansion terms (i.e., $N = 50$) when the scaling factor is used. However, Fig. 4 shows that about 140 expansion terms are required for the conventional expansion (i.e., $\alpha = 1$).

Many practical problems are required to approximate the distribution function of the form $\exp(-pv^2)$ with moderate and large values of p, for example, the heat equations with Gaussian-type initial distribution with small viscosity coefficients, and



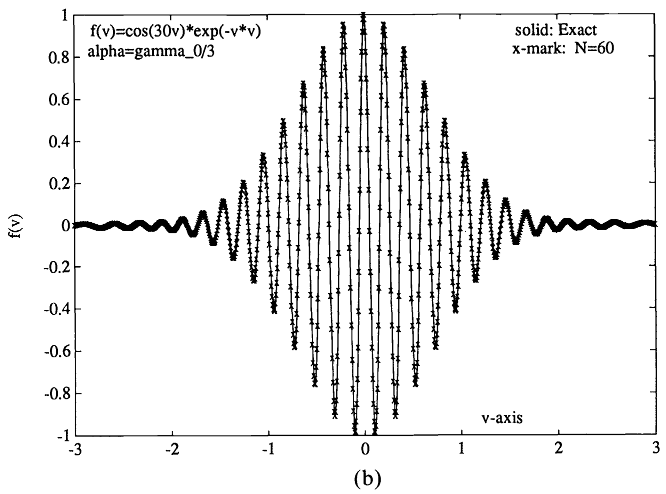


FIG. 1. (a) *Hermite spectral approximation for* $f(v) = \cos(20v)\exp(-v^2)$ *with* $N = 40$ *and* $\alpha = \gamma_0/3$; (b) *Hermite spectral approximation for* $f(v) = \cos(30v)\exp(-v^2)$ *with* $N = 40$ *and* $\alpha = \gamma_0/3$.

the Fokker-Planck equations with small thermal velocity (cf. [17]) or with small particle response time (cf. [10]). Since

$$\exp(-pv^2) = \sum_{n=0}^{\infty} c_{2n}\psi_{2n}(v), \tag{3.13}$$

where

$$c_{2n} = \frac{(-1)^n}{\sqrt{2^{2n}(2n)!(p+\frac{1}{2})}}\left(\frac{p-\frac{1}{2}}{p+\frac{1}{2}}\right)^n \frac{(2n)!}{n!}, \tag{3.14}$$

we obtain the asymptotic behavior of the nth term of the right-hand side of (3.13) as

$$\text{term}_n \sim \frac{1}{\sqrt{n\pi p}}\left(\frac{p-\frac{1}{2}}{p+\frac{1}{2}}\right)^n. \tag{3.15}$$

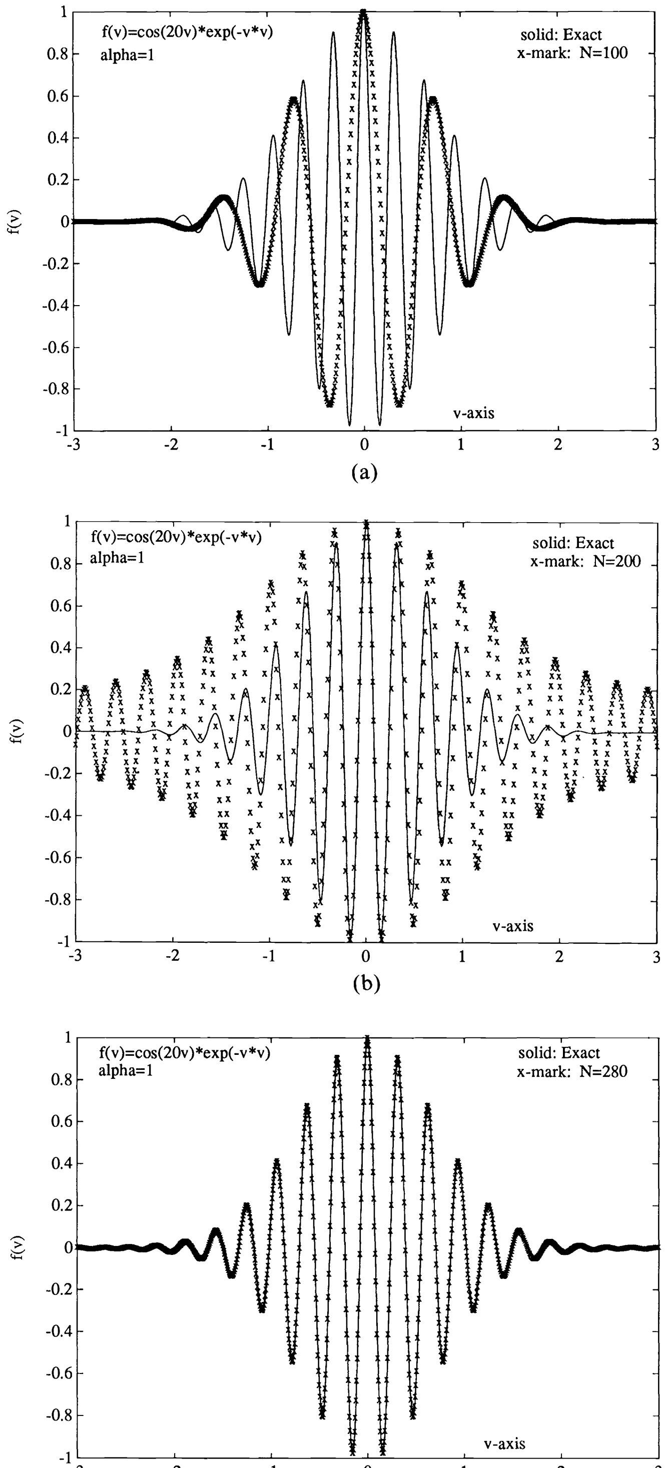
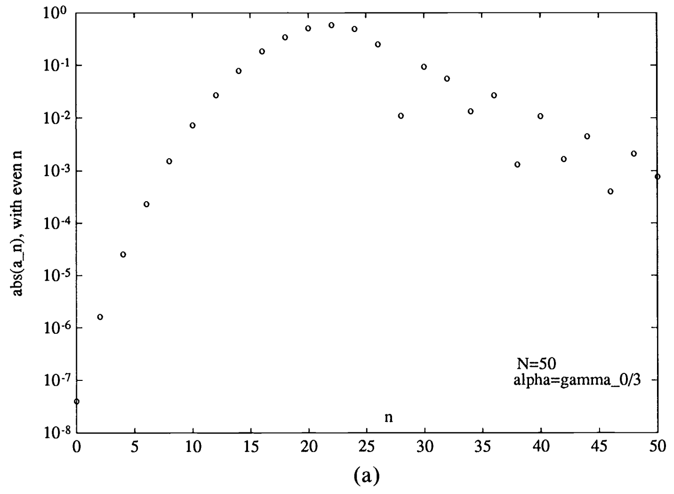


FIG. 2. *Hermite spectral approximation for* $f(v) = \cos(20v)\exp(-v^2)$ *without using a scaling factor.* (a) $N = 100$; (b) $N = 200$; *and* (c) $N = 280$.



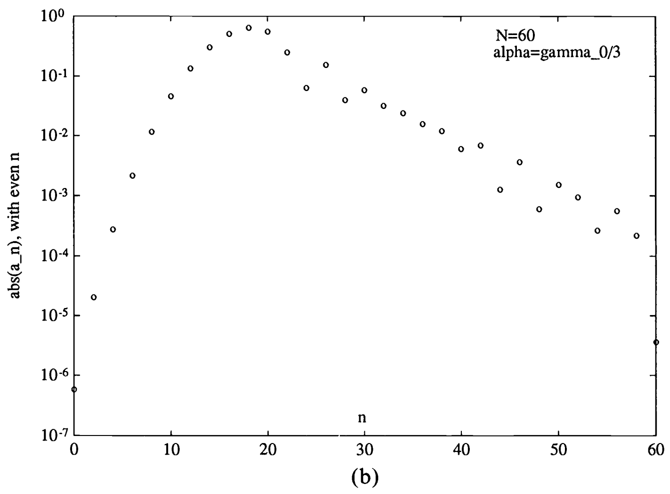


FIG. 3. *The even coefficients of the Hermite expansions for* $f(v)=\cos(20v)\exp(-v^2)$. (a) $N=50$ *and* $\alpha=\gamma_0/3$; (b) $N=60$ *and* $\alpha=\gamma_0/3$.

The above result is bad: for large p about $O(p)$ terms are required. This can be seen from the following. Since

$$\left(1-\frac{1}{x}\right)^x \leqq \lim_{a\to\infty}\left(1-\frac{1}{a}\right)^a = \frac{1}{e}, \qquad \text{for all } x \geqq 1, \tag{3.16}$$

then only when $n \geqq N \approx Cp$ with a positive constant C (which is quite large),

$$\text{term}_n \sim \frac{1}{\sqrt{n\pi p}}\, e^{-C}. \tag{3.17}$$

However, when applying the scaling technique to the same function, we obtain

$$\exp(-pv^2) = \sum_{n=0}^{\infty} d_{2n}\psi_{2n}(\alpha_N v) \tag{3.18}$$

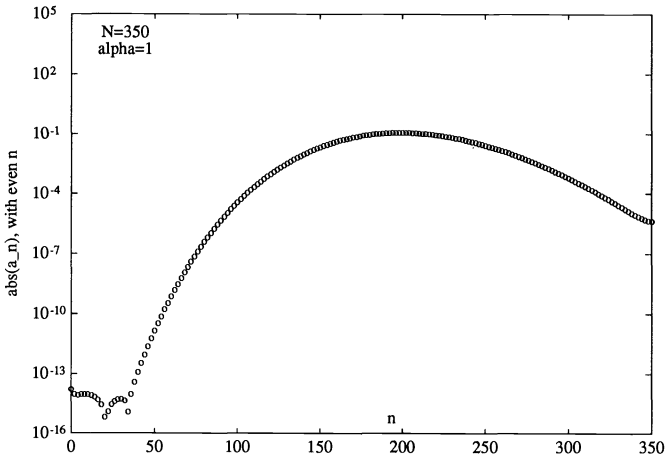


FIG. 4. *The even coefficients of the standard Hermite expansions* (i.e., $\alpha=1$) *for* $f(v)=\cos(20v)\exp(-v^2)$, $N=350$.

with $\alpha_N=\gamma_0/M$, as given in (3.2). The asymptotic form of the nth term is

$$\text{term}_n \sim \frac{\alpha_N}{\sqrt{n\pi p}}\left(\frac{p/\alpha_N^2-\frac{1}{2}}{p/\alpha_N^2+\frac{1}{2}}\right)^n, \tag{3.19}$$

which yields that

$$\text{term}_n \sim \sqrt{\frac{2N}{n\pi p M^2}}\left(\frac{M^2p/N-\frac{1}{2}}{M^2p/N+\frac{1}{2}}\right)^n. \tag{3.20}$$

If p is large, then M can be chosen as one. If $n \geqq N$, the right-hand side of (3.20) decays rapidly to zero when $N \geqq C(p/N+\frac{1}{2})$ with a positive constant C:

$$\begin{aligned}\text{term}_n &\sim \sqrt{\frac{2N}{n\pi p}}\left(1-\frac{1}{p/N+\frac{1}{2}}\right)^n\\ &\leqq \sqrt{\frac{1}{p}}\left(1-\frac{1}{p/N+\frac{1}{2}}\right)^{C(p/N+1/2)}\\ &\leqq \sqrt{\frac{1}{p}}e^{-C}.\end{aligned} \tag{3.21}$$

The requirement $N \geqq C(p/N+\frac{1}{2})$ is satisfied when $N=O(\sqrt{p})$. In Figs. 5 and 6 we consider a test function $f(v)=\exp(-2v^2)+\exp(-20v^2)$. It can be seen from Fig. 5 that after scaling ten expansion terms (again, since the function f is an even function, the number of expansion terms is $N/2$) give a good approximation. However, Fig. 6 shows that approximations without scaling need more than 50 expansion terms.

4. Numerical applications. Consider the eigenvalue problem [3, p. 126]

$$-u''(v)+v^4u(v)=\lambda u(v). \tag{4.1}$$

By the WKB method, the solution of the above equation has the asymptotic behavior

$$u(v)\sim \exp(-|v|^3/3). \tag{4.2}$$

It is obvious from (4.2) that $u\sim 0$ if $|v| \geqq M \approx 5$. In order to obtain accurate solutions of (4.1) efficiently, we need to choose the scaling factor $\alpha=\gamma_0/M$, where $\gamma_0=\max_{0\leqq j\leqq N}\{\gamma_j\}$, with γ_j the roots of $H_{N+1}(\gamma)$. Since the solutions of (4.1) are even

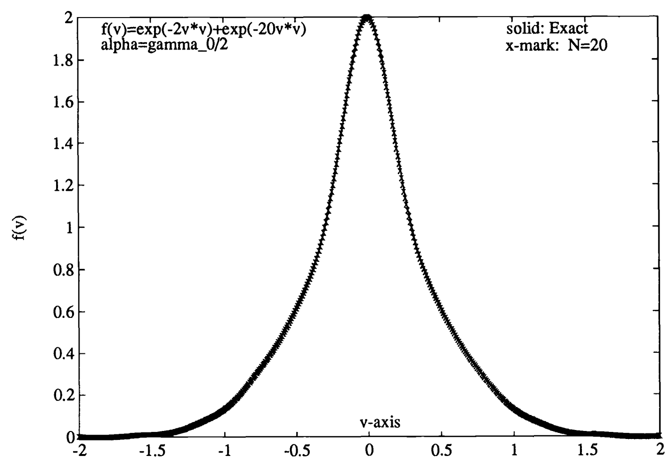


FIG. 5. *Hermite spectral approximation for* $f(v) = \exp(-2v^2) + \exp(-20v^2)$, *with* $\alpha = \gamma_0/2$ *and* $N = 20$.

functions, only $N/2$ expansion terms are required in the actual calculations. For $N = 60$, we predict that the scaling factor $\alpha \approx 10.16/5.0 \approx 2.0$. Birkhoff and Fix [3] used Galerkin's method with 30 Hermite functions (i.e., $N = 60$) to solve (4.1). They found that the standard Hermite functions (i.e., without scaling) gave the first 18 eigenvalues to only three decimal places, whereas using a scaling factor $\alpha = 2.154$ gave the same eigenvalues to 10 decimal places. That is, an increase of 10^{-7} in accuracy is obtained. They obtained the optimum scaling factor through trial and error (the procedure requires a considerable amount of computer time), but the present work provides an accurate scaling factor in a very simple way.

Finally, we apply the scaling technique to the one-dimensional heat equations. The numerical methods for heat equations have been studied extensively in the past (see, e.g., [16] and [18]). Consider the equation

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial v}\left(\nu \frac{\partial u}{\partial v}\right), \tag{4.3}$$

where ν is the viscosity coefficient. If an initial temperature distribution is known, then the problem is to determine the temperature distribution at later times. If the viscosity coefficient is a constant and the initial distribution is given as

$$u(v, 0) = \frac{1}{\sqrt{\nu\pi}} \exp\left(-\frac{v^2}{\nu}\right), \tag{4.4}$$

then the exact solution of (4.3) and (4.4) is

$$u(v, t) = \frac{1}{\sqrt{\pi\nu(4t+1)}} \exp\left(-\frac{v^2}{\nu(4t+1)}\right). \tag{4.5}$$

Problem (4.3), (4.4) has been chosen since it has an analytic solution and this allows us to compare our numerical results with the exact solution (4.5). We use the pseudospectral method introduced in § 2 to compute the numerical solutions. The numerical procedure can be applied to more complicated initial distributions and to variable viscosity coefficients. It can be seen from the previous section that the Hermite spectral methods work well for moderate values of ν, but about $O(1/\nu)$ expansion terms are needed when ν is small. However, if we apply the scaling technique, then fewer terms are required. To illustrate this, we shall consider the case when $\nu = 0.01$.

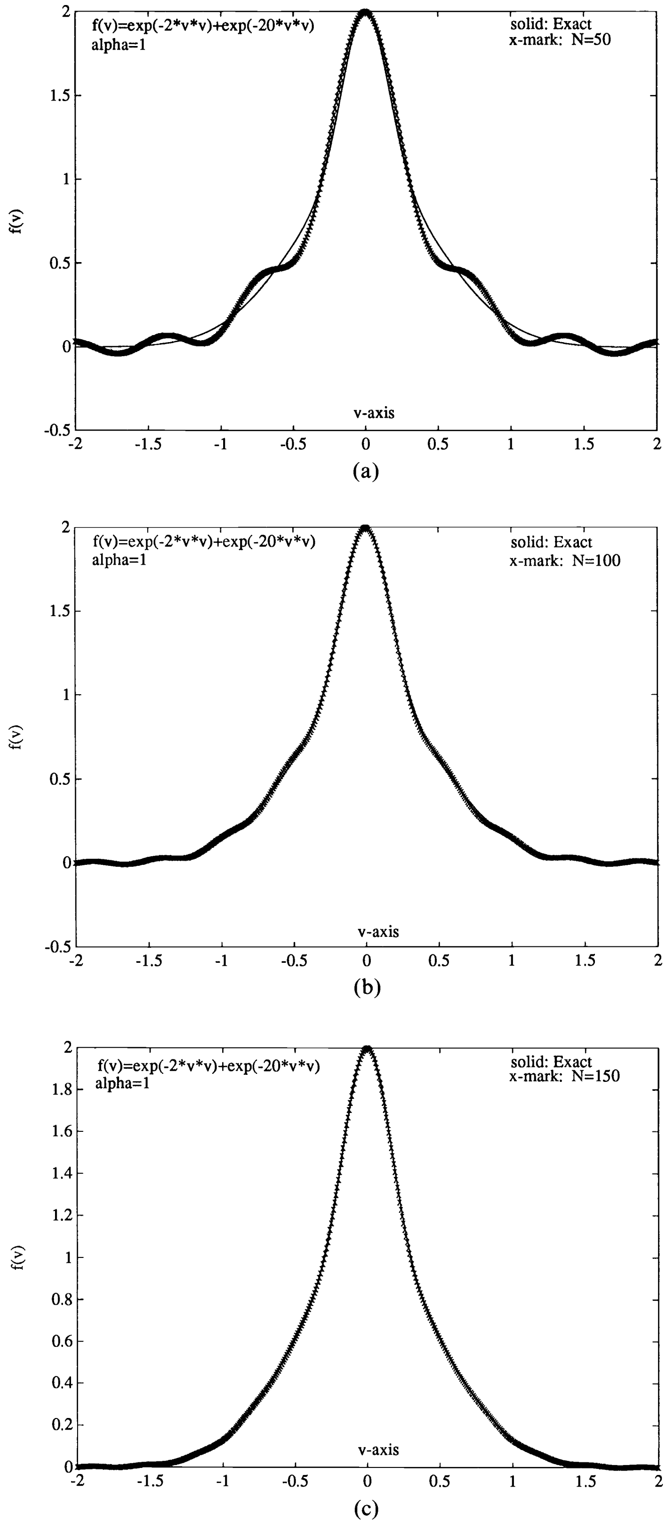


FIG. 6. *Hermite spectral approximation for* $f(v) = \exp(-2v^2) + \exp(-20v^2)$ *without using a scaling factor.* (a) $N = 50$; (b) $N = 100$; *and* (c) $N = 150$.

Let $U(t)=[u(\gamma_0/\alpha_N, t), \ldots, u(\gamma_N/\alpha_N, t)]^T$. Then (4.3) may be semidiscretized as

$$\frac{dU}{dt}=\nu D^{(2)}U, \tag{4.6}$$

where $D^{(2)}$ is the second-order differentiation matrix, which can be computed by (2.4), (2.6), and (2.10). When an explicit method is used to integrate (4.6) in time, the maximum allowable time step needs to satisfy

$$\Delta t=O\left(\frac{1}{\nu sr(D^{(2)})}\right), \tag{4.7}$$

where $sr(D^{(2)})$ denotes the spectral radius of the matrix $D^{(2)}$. Since $sr(D^{(2)})=O(\alpha_N^2 N)$ (see [19]) and $N=O(\sqrt{1/\nu})$ (see § 3), $\alpha_N=O(\sqrt{N})$, we obtain $\Delta t=O(1)$. This suggests that the time-step size can be independent of N when ν is small, which is unlike the finite difference methods [16] and the particle methods [18]. Figure 7 gives a comparison

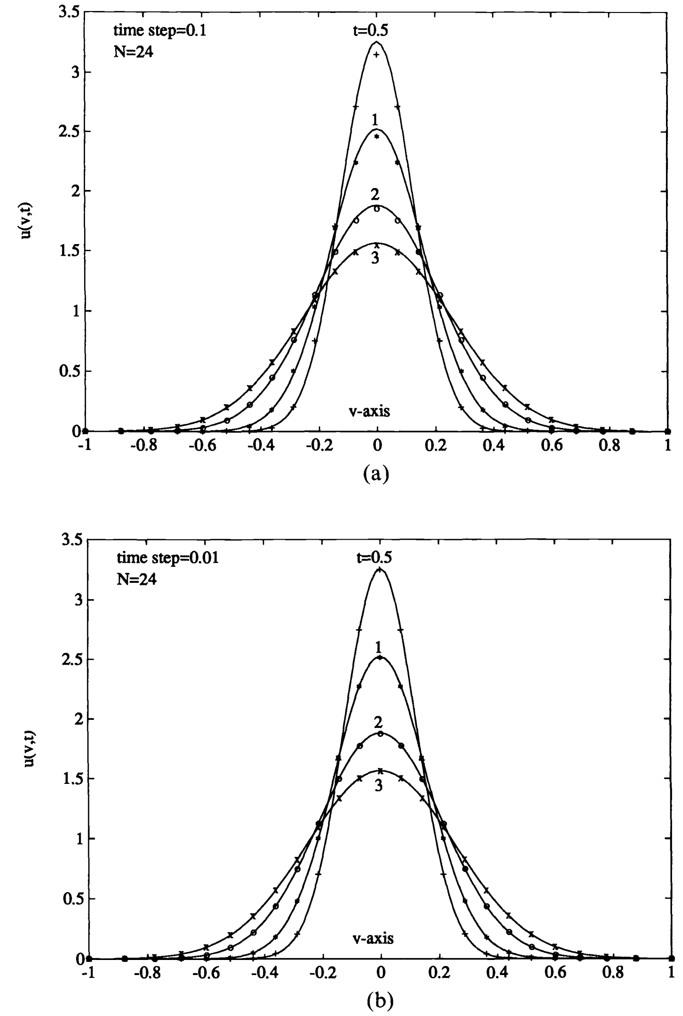


FIG. 7. *Comparison between the exact (lines) and numerical (marks) solution for the diffusion equation at different times.* (a) $\Delta t=0.1$ *and* $N=24$; (b) $\Delta t=0.01$ *and* $N=24$.

between the exact solution and the numerical results. The solution domain is $|v| \leqq 1$ and N is 24 (which corresponds to 12 expansion terms, since the solution is an even function). Figure 7(a) shows that even for a quite large step size, $\Delta t = 0.1$, stable numerical results can be obtained. The results differ only slightly from those obtained by using a smaller time-step size (see Fig. 7(b)). The numerical results are obtained by using the forward Euler method with constant time-step size Δt.

5. Discussions. The method presented in this paper can be used in solving differential equations with solutions that decay exponentially as $v \to \infty$. The idea of using a scaling factor can also be extended to other spectral methods for unbounded intervals, for example, Laguerre spectral methods (see, e.g., [13]) and rational spectral methods (see, e.g., [7] and [19]). It is known that Hermite spectral methods cannot handle functions that decay algebraically with v, but Laguerre and rational spectral methods are appropriate in approximating slowly decaying functions (see, e.g., [5]). It is expected that the use of Laguerre or rational spectral methods with a similar scaling technique can approximate solutions of some practical problems (e.g., problems in [7], [8], and [13]).

Acknowledgments. This research was initiated during my visit to University of Strathclyde, Scotland, in the summer of 1991. The hospitality of Professor Sean McKee is gratefully acknowledged. I wish to thank Mr. Qiang Lan for providing me with Fig. 7. I also wish to thank Professors J. P. Boyd and J. A. C. Weideman for their interest and comments on this work.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1972.

[2] K. Banerjee, *General anharmonic oscillators*, Proc. Roy. Soc. London, A364 (1978), pp. 265-275.

[3] G. Birkhoff and G. Fix, *Accurate eigenvalue computation for elliptic problems*, in Numerical Solution of Field Problems in Continuum Physics, Vol. 2, SIAM-AMS Proceedings, AMS, Providence, RI, 1970, pp. 111-151.

[4] J. P. Boyd, *The rate of convergence of Hermite function series*, Math. Comp., 35 (1980), pp. 1039-1316.

[5] ———, *Chebyshev and Fourier Spectral Methods*, Springer-Verlag, Berlin, 1989.

[6] ———, *Asymptotic coefficients of Hermite function series*, J. Comput. Phys., 54 (1984), pp. 382-410.

[7] ———, *Spectral methods using rational basis functions on an infinite interval*, J. Comput. Phys., 69 (1987), pp. 112-142.

[8] J. P. Boyd and D. W. Moore, *Summability methods for Hermite functions*, Dynam. Atmos. Sci., 10 (1986), pp. 51-62.

[9] H. C. Brinkman, *Brownian motion in a field of force and the diffusion theory of chemical reactions*, Physica, 22 (1956), pp. 29-34.

[10] S. Chandrasekhar, *Stochastic problems in physics and astronomy*, Rev. Modern Phys., 15 (1943), pp. 1-89.

[11] D. Funaro and O. Kavian, *Approximation of some diffusion evolution equations in unbounded domains by Hermite functions*, Math. Comp., 57 (1991), pp. 597-619.

[12] D. Gottlieb and S. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, CBMS-NSF Regional Conf. Series Appl. Math., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[13] Y. Maday, B. Pernaud-Thomas, and H. Vandeven, *Reappraisal of Laguerre type spectral methods*, Rech. Aerospat., 6 (1985), pp. 13-35.

[14] H. G. Marshall and J. P. Boyd, *Solitons in a continuously stratified equatorial ocean*, J. Phys. Oceangr., 17 (1987), pp. 1016-1031.

[15] T. Tang, S. McKee, and M. W. Reeks, *A spectral method for the numerical solutions of a kinetic equation describing the dispersion of small particles in a turbulent flow*, J. Comput. Phys., to appear.

[16] R. D. Richtmyer and K. W. Morton, *Difference Methods for Initial-Value Problems*, 2nd ed., Interscience Publishers, New York, 1967.

[17] H. RISKEN, *The Fokker-Planck Equation: Methods of Solution and Applications*, 2nd ed., Springer-Verlag, Berlin, 1989.

[18] G. RUSSO, *A particle method for collisional kinetic equations.* I. *Basic theory and one-dimensional results*, J. Comput. Phys., 87 (1990), pp. 270-300.

[19] J. A. C. WEIDEMAN, *The eigenvalues of Hermite and rational spectral differentiation matrices*, Numer. Math., 61 (1992), pp. 409-431.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 607–626, May 1993

ACCELERATED MULTIGRID CONVERGENCE AND HIGH-REYNOLDS RECIRCULATING FLOWS*

A. BRANDT† AND I. YAVNEH‡

Abstract. Techniques are developed for accelerating multigrid convergence in general, and for advection-diffusion and incompressible flow problems with small viscosity in particular. It is shown by analysis that the slowing down of convergence is due mainly to poor coarse-grid correction to certain error components, and means for dealing with this problem are suggested, analyzed, and tested by numerical experiments, showing very significant improvement in convergence rates at little cost.

Key words. multigrid, incompressible flow, acceleration

AMS(MOS) subject classifications. 65N20, 76D05

1. Introduction.

Classical multigrid methods were originally developed for elliptic partial differential equations and systems. For such problems these methods have proved to be extremely efficient, enabling solution to the level of discretization errors in just a few minimal work units, so that the total work invested in the solution grows linearly with the number of variables, and usually at most several dozen operations per variable are required. When applied to nonelliptic and singular perturbation problems such as high-Reynolds flows, however, performance seems to deteriorate significantly, due in part to the fact that the solutions become more complex (e.g., boundary layers and characteristic directions along which high-frequency data may propagate to large distances). But poor multigrid behavior is exhibited even in simple problems with smooth solutions when the partial differential operator is nonelliptic (or has a nonelliptic component).

To learn how to treat the various troubles that appear in flow problems, it is necessary to distinguish them. One of the main distinctions that must be made is between *entering flows*, in which the flow enters through some boundary and follows a well-defined general orientation, and *recirculating* flows. In the former, relaxation can be made to resolve smooth components, and thus act (in part) as a *solver* and not just as a *smoother* in the multigrid solution process. Highly efficient multigrid solutions to problems of this type are demonstrated in [7]. The purpose of the present study is to show how to improve performance greatly in recirculating flows and other problems, in which relaxation significantly resolves only nonsmooth error components.

The main tool used in analysis and prediction of the performance of multigrid algorithms is local mode analysis (see, e.g., [2]). For elliptic partial differential systems it was shown in [4] that, under reasonable assumptions, the predicted performance can indeed be obtained for general domains by adding some processing, for negligible cost, at and near the boundaries. Hence, boundary effects may be neglected in the basic

*Received by the editors March 11, 1991; accepted for publication (in revised form) April 20, 1992. This research was supported in part by the Air Force Office of Scientific Research, United States Air Force, under grant AFOSR-91-0156, by the National Science Foundation under grant NSF DMS-9015259, and by the Department of Energy under contracts W-7405-ENG-36 and KC-07-01-01, while the second author was visiting the Center for Nonlinear Studies in the Los Alamos National Laboratory.

†Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel.

‡Oceanography Department, National Center for Atmospheric Research, P.O. Box 3000, Boulder, Colorado 80307-3000. This research was partly performed while the author was at the Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel.

analysis of multigrid solvers for elliptic problems. This is not the case in problems that are nonelliptic or have nonelliptic components, such as incompressible flows at high-Reynolds numbers. In such problems, high-frequency boundary data may propagate far into the domain, and it may be necessary to include boundary effects in the analysis (see [7]). However, when dealing with shear-driven recirculating flows we are once again in a position that is similar to the elliptic case with respect to boundary effects, since cross-stream behavior is determined by the (elliptic) viscosity terms no matter how small they may be (see [6]). For such flows the infinite-space local mode analysis is again useful.

Slow convergence of multigrid cycles can generally be traced to (at least) one of two major causes: poor smoothing or poor coarse-grid approximation to the fine-grid problem. Smoothing, which is reducing the amplitude of high-frequency errors that cannot be treated on the next coarser grid, is dealt with extensively in most publications on multigrid solvers for incompressible flows (see, e.g., [2] and [5]). Here we address the problem of poor coarse-grid corrections, which may require specialized methods that depend on the cause of the problem. When the poor approximation is caused by a local factor, say, some singularity in the boundary, the proper course is usually to employ local methods such as extra relaxation in the vicinity of the singularity (see [1] and [4]). But sometimes, in particular in high-Reynolds flows, the coarse grid fails to approximate the fine-grid problem well enough for certain components *throughout the domain.* Frequently, the best course of action then is to simply disregard this seemingly poor behavior, since those components, which are poorly approximated by the coarse grid, do not in turn approximate the differential solution well (see [2] and [3]). If, however, one is interested in obtaining good *algebraic* convergence and not just rapid convergence to a good approximation of the *differential* solution, special techniques must be employed to accelerate convergence.

Consider as our model problem the constant-coefficient advection-diffusion equation:

$$L_{ad}u = -\epsilon\triangle u + au_x = 0\,, \tag{1}$$

where $\triangle$ is the Laplacian operator and ϵ is a positive constant. Suppose that this equation is discretized by some finite difference scheme L^h_{ad} of order q on a uniform grid of mesh-size h whose orientation is general (and therefore choosing x to be the characteristic direction results in no loss of generality). The resulting discrete set of equations is solved by starting with some initial approximation to the discrete solution on the fine grid h (grid with mesh-size h) and iterating with the usual multigrid cycles. Soon one finds that (for vanishing ϵ) the residual norms are reduced at best by an amplification factor of $1 - 0.5^q$ by each cycle, even if the number of relaxation sweeps per level and the cycle index γ (defined below) are chosen to be quite large. The reason for this slow-down has been shown to be poor approximation of *smooth characteristic components* by the coarse grids. This property has already been explained in [3]. We return to it here, and in another context in [7], and present methods for treating the problem.

As usual, when researching the advection-diffusion equation, our object is to learn how to treat the Navier–Stokes equations. Since these exhibit similar behavior, we conclude that the problem of poor coarse-grid correction studied here is the main cause for the poor convergence rates of flow problems as well. Hence, although the analyses below are all done for the advection-diffusion equation, numerical experiments also include the incompressible flow equations, and indeed the behavior is seen to be influenced similarly by the methods proposed.

In §§2 and 3 the two-level and multilevel cycles are analyzed, and it is shown why the usual multigrid cycles exhibit poor convergence rates. In §4 it is shown by analysis and numerical examples that significant acceleration of convergence can be obtained at virtually no cost by overweighting some of the residuals that are transferred to the next coarser grid. In §5 a method of employing defect corrections within the multigrid cycle is introduced, and it is shown by analysis and numerical experiments that this method can be combined in various ways with residual overweighting to greatly improve the multigrid cycle's performance. The ultimate modified W cycle that is developed enables an asymptotic reduction of the error by a factor of nine per cycle in its two-level version for the first-order discretized advection-diffusion and incompressible Navier–Stokes equations, rather than the factor of two that is yielded by the usual two-level cycle. This performance is almost matched in the multilevel cycle as well, and here the improvement is still more dramatic, since the usual cycle's performance is shown to deteriorate rapidly as the number of levels grows. Conclusions and remarks are given in §6.

The accelerated multigrid analyses and methods developed in this article can be straightforwardly generalized to other cases in which multigrid convergence deteriorates due to poor coarse-grid corrections for some smooth components.

2. Two-level infinite-space local mode analysis. In order to have a clear and quantitative understanding of the slowing down of multigrid convergence due to poor coarse-grid correction, we will analyze here a multigrid cycle leaving out some irrelevant aspects: we will treat only smooth components, and hence neglect the effect of intergrid transfers, and employ the first differential approximation (FDA; see [11], [3], and [2, §7.5]) to the difference equations. We will also assume for simplicity that the diffusion coefficient ϵ tends to zero, which is indeed the case for which the multigrid performance is usually the worst. The FDA approximation to a qth-order discretization of L_{ad} then has the form

$$L_{ad}^h = -\epsilon_h a T_{q+1}(\partial_x, \partial_y) + a\partial_x \,, \tag{2}$$

where the first term on the right-hand side represents the first truncated term in the discretization of L_{ad}. T_{q+1} is a polynomial of degree $q+1$ of the form

$$T_{q+1}(\partial_x, \partial_y) = \sum_{j=0}^{q+1} a_j \, \partial_x^j \, \partial_y^{q+1-j}, \tag{3}$$

and $\epsilon_h = O(h^q)$. ϵ_h and the coefficients a_j are determined by the particular discretization and angle between the characteristic direction and the grid. For example, in the case of the usual first-order upstream discretization, the FDA is

$$\begin{aligned} L_{ad}^h = -\tfrac{1}{2}ha[\,&(\cos^3\phi + \sin^3\phi)\partial_{xx} + \sin 2\phi(\sin\phi - \cos\phi)\partial_{xy} \\ &+ \tfrac{1}{2}\sin 2\phi(\sin\phi + \cos\phi)\partial_{yy}\,] + a\partial_x \,, \end{aligned}$$

where ϕ is the angle that the characteristic direction x forms with the grid. (This is obtained by writing the artificial viscosity of upstream discretization in the coordinates of the grid, and transforming to the characteristic coordinates x and y.)

Consider an error component on the fine grid h given by

$$v^h = e^{i(\omega_1 x + \omega_2 y)} \,. \tag{4}$$

Generalization to higher dimensions is straightforward, and the following analyses apply directly. The residual due to this error is

$$r^h = -L^h_{ad}v^h = -\hat{L}^h_{ad}(\omega_1, \omega_2) \cdot e^{i(\omega_1 x + \omega_2 y)}\,, \tag{5}$$

where $\hat{L}^h_{ad}$ is called the *symbol* of L^h_{ad} (see also [2]). The correction problem solved on the coarse grid H is

$$L^H_{ad}v^H = r^h\,, \tag{6}$$

yielding

$$v^H = -\frac{\hat{L}^h_{ad}}{\hat{L}^H_{ad}}v^h\,. \tag{7}$$

Denoting

$$C(\omega_1, \omega_2) = \frac{\hat{L}^h_{ad}(\omega_1, \omega_2)}{\hat{L}^H_{ad}(\omega_1, \omega_2)}\,,$$

we obtain that the two-level amplification factor ν of the fine-grid error, defined as the ratio of the new fine-grid error (after the correction v^H has been added) to the original error v^h, is given by

$$\nu(\omega_1, \omega_2) = \frac{v^h + v^H}{v^h} = 1 - C(\omega_1, \omega_2)\,. \tag{8}$$

From (2), (3), and (8), for a qth-order discretization of the advection-diffusion operator, we obtain

$$\nu(\omega_1, \omega_2) = 1 - \frac{\epsilon_h T_{q+1}(i\omega_1, i\omega_2) + i\omega_1}{\epsilon_H T_{q+1}(i\omega_1, i\omega_2) + i\omega_1}\,. \tag{9}$$

For most smooth components (and small mesh-sizes) this amplification factor is close to zero, and multigrid convergence can hence be expected to be essentially as good as allowed by the smoothing rate and the number of relaxation sweeps per level. But consider smooth *characteristic error components*, which are smooth (and therefore nearly unaffected by any local-type relaxation), but much smoother in the characteristic direction x than in the cross-characteristic direction y. For such components, $\omega_1/\omega_2 \ll 1$, and their amplification factor is therefore much larger. In the limit of components which only vary in the cross-characteristic direction, we obtain that the two-level *convergence factor* ν_{tl}, which is defined as the maximal (in absolute value) ν over all the frequencies defined on the grid, is given by

$$\nu_{tl} = \nu(0, \omega_2) = 1 - \frac{\epsilon_h}{\epsilon_H} \approx 1 - 0.5^q\,. \tag{10}$$

Asymptotically the poorly corrected characteristic components become dominant, and the error norm is reduced at best by a factor of $1 - 0.5^q$, even with two-level cycles.

3. Multilevel local mode analysis. The multigrid *cycle index* γ is defined as the number of times a correction from the next coarser grid is taken by each intermediate level in calculating its solution, before that solution is used to correct the error on the next finer grid. Thus $\gamma = 1$ is the index of the so-called V cycle, and $\gamma = 2$ is the index of the W cycle. Actual multigrid cycles need to employ a fairly small cycle index γ if the total work is to grow only linearly with the number of variables: in d dimensions, γ must be smaller than 2^d. Let us therefore determine by analysis the expected multilevel amplification factor of an error component whose two-level amplification factor $1 - C$ is assumed to be independent of the mesh-size. This simplifying assumption is only valid for the extreme cases where, in (9), ω_1 vanishes, in which case C equals $\epsilon_h/\epsilon_{2h} \approx 0.5^q$, or where T_{q+1} vanishes, in which case C equals one. This is appropriate here since these extreme cases determine the worst-case amplification factor. Moreover, this analysis of the multigrid cycle is not restricted to the advection-diffusion equation.

Consider a multigrid cycle that is implemented on $n+1$ levels numbered $0, \dots, n$, with n denoting the finest grid. For each error component there corresponds a "solution level" i on which it is sufficiently nonsmooth so as to be eliminated efficiently by relaxation (or perhaps direct solution in the case $i = 0$). To simplify, we assume that every error component is eliminated completely on its solution level, but that on all finer grids it is unaffected by relaxation. Let us denote by $\nu_\gamma^{n-i}(C)$ the level-n amplification factor of an error component, whose two-level amplification factor is $1 - C$ and whose solution level is i, in a multigrid cycle with cycle index γ. Let $k = n - i$. Under the assumptions above, we have for $k = 0$

$$\nu_\gamma^0 = 0\,, \tag{11}$$

and for $k = 1$

$$\nu_\gamma^1 = 1 - C, \tag{12}$$

regardless of γ (the latter is simply the two-level cycle by definition). For $1 < k \le n$ the residual problem is transferred to the next coarser grid (level $k-1$), and γ multigrid cycles (in which the corresponding amplification factor for that component is ν_γ^{k-1}) are performed. As a result, the error in the solution to the coarse-grid problem is reduced by a factor of $(\nu_\gamma^{k-1})^\gamma$. This approximate correction is now transferred back to the fine grid, but multiplied by C (since even the exact solution to the coarse-grid problem only yields C times the required correction). This yields the following recurrence equation:

$$\nu_\gamma^k = 1 - C \cdot \left[1 - (\nu_\gamma^{k-1})^\gamma\right] = \nu_\gamma^1 + C(\nu_\gamma^{k-1})^\gamma\,, \qquad 1 \le k \le n\,. \tag{13}$$

3.1. Properties of the multilevel cycle. In the following discussion we consider only real values of C, since C is real in both extreme cases of the advection-diffusion equation: when the coarse-grid correction is worst, and when it is best. This assumption will greatly simplify the discussion. Also, it is of course unnecessary to consider nonpositive real values for C, since these would imply that the coarse grid does not approximate the corresponding components at all, in which case special measures need to be taken.

Remark 1. The fixed points of (13) are solutions of

$$\nu = 1 - C(1 - \nu^\gamma)\,. \tag{14}$$

$\nu = 1$ is a fixed point for every γ. When $C < 1$ it is the only one in the case of a V cycle ($\gamma = 1$). *W* cycles ($\gamma = 2$) have a second fixed point at $\nu = (1 - C)/C$.

PROPOSITION 1. *For every $k \geq 1$,*

$$\nu_1^k = 1 - C^k . \tag{15}$$

Proof. By induction: $\nu_1^1 = 1 - C$, and by (13) and the induction hypothesis,

$$\nu_1^k = 1 - C + C(1 - C^{k-1}) = 1 - C^k . \qquad \square$$

Conclusion. The convergence rate of the multigrid V cycle deteriorates rapidly as the number of levels grows, unless C is close to 1. In particular, the V cycle is clearly unsuitable for the advection-diffusion equation, even with first-order discretization.

PROPOSITION 2. *For all $0 < C < 1$ and all positive k and γ,*

$$0 \leq \nu_\gamma^{k-1} < \nu_\gamma^k < 1. \tag{16}$$

Proof. By induction: $0 < \nu_\gamma^1 = 1 - C < 1$, and by (13) and the induction hypothesis,

$$\nu_\gamma^{k+1} = 1 - C \cdot [1 - (\nu_\gamma^k)^\gamma] < 1 \tag{17}$$

and

$$\nu_\gamma^{k+1} - \nu_\gamma^k = C \cdot \left[(\nu_\gamma^k)^\gamma - (\nu_\gamma^{k-1})^\gamma\right] > 0 . \tag{18} \qquad \square$$

PROPOSITION 3. *For all $0 < C < 1$, ν_2^k tends to the smaller of the two fixed points (1 and $(1 - C)/C$) as k tends to infinity.*

Proof. For $k \geq 0$, by (16) we have

$$\begin{aligned} \frac{\left|\frac{1-C}{C} - \nu_2^{k+1}\right|}{\left|\frac{1-C}{C} - \nu_2^k\right|} &= \frac{\left|\frac{1-C}{C} - 1 + C \cdot [1 - (\nu_2^k)^2]\right|}{\left|\frac{1-C}{C} - \nu_2^k\right|} \\ &= \frac{\frac{1}{C}\left|(1-C)^2 - (C\nu_2^k)^2\right|}{\frac{1}{C}\left|1 - C - C\nu_2^k\right|} = \left|1 - C(1 - \nu_2^k)\right| < 1 . \end{aligned} \tag{19}$$

The proof follows from this and Proposition 2. $\square$

Conclusion. Proposition 3 implies that ν_2^k tends to one when $0 < C \leq 0.5$. Thus, the convergence factor of the W cycle for the advection-diffusion equation with first-order discretization tends to one as the number of levels tends to infinity.

We reiterate that this analysis assumes that relaxation is of a local type, and therefore has a negligible effect on all components that are smooth on the scale of the grid on which they are relaxed. A different situation may arise in cases such as *entering flow* problems, which are studied in [7]. When relaxation is performed in downstream ordering in such problems, it no longer has a purely local effect. Indeed, for the advection equation with upstream differencing, if relaxation is carried out in downstream ordering, it performs as a *solver* and not just as a *smoother*. Convergence of the advection-diffusion equation is discussed in [8], and much better rates than the ones predicted by our analysis are obtained. But the model problems are precisely of the entering flow type, and at least some of the relaxation sweeps are carried out in downstream ordering. This point, however, which explains a number of phenomena in the numerical results that are obtained there as well as in other publications, and which is the key to further achievements with much more complicated schemes, seems to have been overlooked. Also, in some flow problems, the actual Reynolds number in the regions of recirculation are very much smaller than the nominal values (because the velocities in these regions are small), and somewhat better performance may then be found (e.g., in [10]).

4. Method of overweighted residuals. A very simple and almost cost-free approach to accelerating convergence is the method of overweighted residuals (OWR). The idea, which is reminiscent of the method of successive overrelaxation (SOR), is to improve the coarse-grid correction to the error in the fine-grid approximation by multiplying the residuals that are transferred to the coarse grid by some constant η between one and two. Clearly, η should not be too large, since those components that usually receive the proper correction are now overcorrected. It is best to determine an optimal overweighting factor η, which will depend on the number of levels employed.

4.1. Two-level single-parameter optimization. Let ξ denote the minimal C over all the frequencies defined on the fine grid (for the advection-diffusion case, $\xi = 0.5^q$). Then the poorest (largest) corresponding error-amplification factor in the usual two-level cycle (with $\eta = 1$) is $1 - \xi$, whereas the best amplification factor is zero. When the residuals are multiplied by the factor η, however, the two extreme error-amplification factors are now given by $1 - \eta\xi$ and $1 - \eta$ (by (6–8)). Due to the monotonicity of these terms, the optimal two-level overweighting factor η_{tl} is obtained when

$$|1 - \eta_{tl}\xi| = |1 - \eta_{tl}|, \tag{20}$$

and therefore

$$\eta_{tl} = \frac{2}{1+\xi}. \tag{21}$$

The two-level convergence factor ν_{tl} decreases accordingly from $1 - \xi$ to

$$\nu_{tl} = 1 - \eta_{tl}\xi = \frac{1-\xi}{1+\xi}. \tag{22}$$

The two-level convergence factor of the advection-diffusion equation (with vanishing diffusion coefficient) employing first-order discretization, for example, improves from 0.5 to 0.33 with $\eta_{tl} = 4/3$.

4.2. Multilevel single-parameter optimization. Propositions 2 and 3 imply that the components that converge most slowly, for a fixed C in (0,1), are those whose solution level is the lowest ($k = n$). When the number of levels is large, the amplification factor of such errors by a W cycle tends to $(1 - C)/C$ when C is between one half and one. For the *overcorrected* components in a cycle with an even cycle index γ, however, we have the following.

PROPOSITION 4. *For all* $1 < C < 2$, *even* γ, *and* $k \geq 1$,

$$|\nu_\gamma^k| \leq |\nu_\gamma^1| = C - 1. \tag{23}$$

Proof. By induction: For $k \geq 1$

$$C - 1 - |\nu_\gamma^{k+1}| = C - 1 - |1 - C + C(\nu_\gamma^k)^\gamma|, \tag{24}$$

but

$$C - 1 + 1 - C + C(\nu_\gamma^k)^\gamma > 0\,, \tag{25}$$

and alternatively

$$C - 1 - (1 - C + C(\nu_\gamma^k)^\gamma) \geq 2(C-1) - C(C-1)^\gamma > 0\,, \tag{26}$$

where in (26) the first inequality is trivial for $k = 1$ and is due to the induction hypothesis for $k > 1$, and the second is due to the range of C. □

Conclusion. The components that converge most slowly when $1 < C < 2$ are those that are corrected on the *second-finest* level ($k = 1$), and the convergence factor for such overcorrected components is at least as good as the two-level rate when γ is even. This is true in particular for the W cycle ($\gamma = 2$). Hence, the optimal multilevel overweighting factor η_{ml} for the W cycle is obtained when

$$\left|\frac{1 - \eta_{ml}\xi}{\eta_{ml}\xi}\right| = |1 - \eta_{ml}|\,, \tag{27}$$

yielding

$$\eta_{ml} = \xi^{-1/2}\,. \tag{28}$$

This value is valid only when $\eta_{ml} < 2$. For $\xi \leq 0.25$ the multilevel convergence factor must tend to one as the number of levels grows. The W-cycle multilevel convergence factor ν_{ml} of the advection-diffusion equation with vanishing diffusion coefficient and first-order discretization thus improves from 1 to $\sqrt{2} - 1 \approx 0.41$, with the optimal overweighting factor $\eta_{ml} = \sqrt{2}$.

With a finite number of levels, the optimal overweighting factor is reduced somewhat, yielding slightly better convergence rates. It is thus possible to calculate the optimal overweighting factors for each level. These start with 4/3 for the second-coarsest grid and increase until they tend to $\sqrt{2}$ as the grid becomes finer. However, there is not much practical gain in doing such careful optimization, since on very coarse grids the assumptions made in the analysis are rather poor anyway, and also the optimal η quickly tends to $\sqrt{2}$. *One can thus uniformly use the overweighting parameter $\eta = \sqrt{2}$.*

4.3. Two-level multiparameter optimization. In the optimizations described above we have considered only a single cycle, and therefore only a single overweighting parameter η. But since we are dealing with asymptotic convergence rates, it is implied that several cycles are performed. Also, since (at least) W cycles need to be employed, more than one cycle per level is performed. This suggests using several different overweighting factors in order to further reduce the error-amplification factor. This is a special case of *polynomial acceleration*, and the optimal choice of overweighting factors, which is calculated with the aid of *Chebyshev polynomials*, yields *optimal Chebyshev acceleration* (see, e.g., [9]).

Consider a smooth error component for which the ratio between the fine-grid and coarse-grid symbols is C, $\xi \leq C \leq 1$. Suppose that m two-level cycles are performed, where the residuals in the jth cycle, $j = 1, \ldots, m$, are overweighted by a factor of η_j. The factor by which the error is amplified in this process is given by

$$[\nu^m(C)]^m = \prod_{j=1}^{m}(1 - \eta_j C)\,, \tag{29}$$

where $\nu^m(C)$ is the average amplification factor per cycle. Assuming, as above, that the extreme eigenvalues of the two-level operator are 0 and $1-\xi$, the optimal polynomial $\nu_{tl}^m(C)$, i.e., the one for which the maximal absolute value over all C's in the interval is minimized, is given by

$$\nu_{tl}^m(C) = \left[\frac{\mathcal{T}_m\left(\frac{1+\xi-2C}{1-\xi}\right)}{\mathcal{T}_m\left(\frac{1+\xi}{1-\xi}\right)}\right]^{1/m}, \tag{30}$$

where $\mathcal{T}_m$ is the Chebyshev polynomial of degree m, given by

$$\mathcal{T}_m(x) = \begin{cases} \cos(m\cos^{-1}x), & -1 \le x \le 1, \\ \cosh(m\cosh^{-1}x), & 1 < x \end{cases} \tag{31}$$

(see [9] for the general development). By (29) the optimal η_j's are the inverses of the zeros of ν_{tl}^m, which can easily be calculated from (31). For $m=1$ the optimal overweighting factor is indeed as in (21), and for $m=2$ the two optimal factors are given by

$$\eta_j = \frac{2}{1+\xi \pm \frac{1-\xi}{\sqrt{2}}}, \qquad j = 1,2. \tag{32}$$

The average amplification factor per cycle of the error is given by

$$\nu_{tl}^m(1) = \left[\mathcal{T}_m\left(\frac{1+\xi}{1-\xi}\right)\right]^{-1/m}. \tag{33}$$

For the advection-diffusion equation with first-order discretization ($\xi = 0.5$), the single-cycle optimization ($m=1$) thus yields a two-level convergence factor of 0.33 per cycle as shown above. Two-cycle optimization ($m=2$) yields an average two-level convergence factor of 0.24 with the optimal overweighting factors of 1.079 and 1.745, and the asymptotic (infinite number of cycles) average two-level convergence factor with optimal overweighting factors is 0.17. With second-order discretization ($\xi = 0.25$) the single-cycle optimization yields a two-level convergence factor of 0.60 per cycle with $\eta = 1.60$. Two-cycle optimization yields an average two-level convergence factor of 0.47 with the optimal overweighting factors of 1.123 and 2.779, and the asymptotic average two-level convergence factor with optimal overweighting is 0.33. However, large overweighting factors are unlikely to be useful in practice, if only because the corresponding amplification of nonsmooth error components means that much better smoothing is then required.

An automatic acceleration method, which would not require a priori analysis, is conceivably useful. However, since the ultimate goal is to employ at most one or two cycles per level in the solution process, methods whose usefulness relies on the execution of many cycles are unlikely to be truly efficient.

4.4. γ-cycle optimization. In a multigrid cycle with cycle index γ there are γ overweighting factors to be chosen for the γ visits to the next coarser grid at each level. Consider again a smooth error component for which the ratio between the fine-grid and coarse-grid symbols is C, $\xi \le C \le 1$. Then, if the overweighting factors η_j are chosen independently of the level, the error amplification factor *of γ cycles* on level $i+k$ is given by the recurrence equation

$$(\nu_\gamma^k)^\gamma = \prod_{j=1}^{\gamma} \left[1 - \eta_j C\left(1 - (\nu_\gamma^{k-1})^\gamma\right)\right] , \tag{34}$$

with

$$\nu_\gamma^0 = 0 . \tag{35}$$

Again a recurrence equation is obtained, for which 1 is a fixed point. The optimization of the η_j's generally needs to be done numerically. The optimal overweighting factors for the W cycle with $\xi = 0.5$ (e.g., for first-order approximations to the advection equation) are 1.085 and 1.843, yielding a convergence factor per single W cycle of 0.27, whereas the single-parameter optimization yields 0.41.

4.5. Numerical experiments. Numerical experiments were carried out with the advection-diffusion equation, discretized by second-order central differences with an added first-order isotropic artificial viscosity term discretized by the five-point star (the importance of the isotropy of the viscosity coefficients is elaborated upon in [6]). The problem solved was

$$-\epsilon \Delta u + \sin(\pi y) \cos(\pi x)\, u_x - \cos(\pi y) \sin(\pi x)\, u_y = 0 ,$$

over the unit square. The results reported here were obtained with Dirichlet boundary conditions and exact solution zero, in order to allow a very large number of cycles without encountering roundoff errors. The algorithms were tested with smooth nonzero solutions as well, and nearly identical performance was observed until (double-precision) roundoff errors were encountered. Since the physical viscosity coefficient ϵ was taken to be zero, the coefficients of the equation are very small at and near the stagnation point $(\frac{1}{2}, \frac{1}{2})$. When residuals are transferred with some averaging, such as the usual full-weighting that was used in these calculations, the right-hand side on the coarse grid in these regions may be much larger than the coefficients, resulting in reduced performance or even instability. This is best overcome by using averaged coefficients in the calculation of the artificial viscosity. Here we determined the artificial viscosity by adding the absolute value of the coefficient at the point of discretization with weight $\frac{1}{2}$ to those of the four nearest neighbors with weights $\frac{1}{8}$, and multiplying this weighted average by $\frac{h}{2}$ as usual. This averaging introduces only an $O(h^3)$ change in the usual artificial viscosity.

The asymptotic convergence factors of the dynamic residuals with various overweighting factors η are presented in Table 1 along with the predictions of the analyses presented above. The fine mesh-size for the two-level results is 1/64. The multilevel results were obtained with four levels, the finest mesh-size being 1/128, and the analytical prediction refers to a four-level (not infinite-level) cycle, and is calculated from (13). The optimal η is then 1.40—slightly smaller than the infinite-level optimum of $\sqrt{2}$.

The multilevel results were calculated with a W(2,1) cycle (a W cycle with two pre- and one postrelaxation per level), and it was verified that increasing the number of relaxation sweeps per level results in only a negligible improvement in the performance, so that indeed three sweeps per level reduce the high-frequency error components sufficiently, and the convergence rate is determined by the coarse-grid correction.

TABLE 1
Asymptotic residual convergence factors of numerical calculations for the advection-diffusion (AD) *equation and incompressible Navier–Stokes* (INS) *equations are compared with analytical predictions for various single overweighting factors η. Two pre- and one postrelaxation sweeps were employed.*

η	Two-level			Multilevel		
	Analytical prediction	Numerical AD	Numerical INS	Analytical prediction	Numerical AD	Numerical INS
1.00	0.50	0.50	0.49	0.69	0.68	0.66
1.10	0.45	0.45	0.45	0.62	0.61	0.61
1.20	0.40	0.39	0.40	0.55	0.54	0.54
1.30	0.35	0.34	0.33	0.47	0.46	0.45
1.33	0.33	0.33	0.33	0.44	0.44	0.46
1.40	0.40	0.38	0.40	0.40	0.40	0.51
1.50	0.50	0.45	0.51	0.50	0.50	0.63

The experiments were repeated with the incompressible Navier–Stokes (INS) equations in two dimensions, over the unit square with a square of side 0.25 removed from its center. Both the inner and outer squares' sides were aligned with the grid. Dirichlet boundary conditions for the velocities were specified at the inner and outer boundaries. The velocities normal to the boundaries were all made to vanish. The tangential velocities at the inner boundaries were set to zero as well, but the tangential velocities at the outer boundaries were prescribed to be

$$U_{\tan} = \sin \pi s\,,$$

with s varying from 0 to 1 along each side of the outer square, and $U_{\tan}$ driving the flow in the clockwise direction. These conditions and this domain yield a smooth flow with closed streamlines and almost no boundary layers. We chose such a flow because boundary layers constitute a separate problem that needs to be treated by its own specialized methods. Slow convergence due to poor resolution on coarse grids is unrelated to the problem of poor convergence of certain *smooth* components, which is examined here.

The cycle parameters chosen were the same as for the advection-diffusion (AD) equation tests. The discretization used in [5] and [2] was employed with first-order isotropic artificial viscosity, and the Reynolds number solved with was 10^{-6}, so that the physical viscosity was everywhere negligible relative to the artificial viscosity. This is implied by the analysis (and verified by experiments) to be the most difficult case with respect to rate of convergence. Adding physical viscosity, while fixing the overweighting factors, *always* resulted in improved convergence rates.

Now the number of cycles performed was limited by the double-precision roundoff errors. Distributive Gauss–Seidel relaxation with red-black ordering (see [2]) was used throughout. These results are also listed in Table 1 and compared with the analytical prediction for the AD equation.

The numerical results match the analytical prediction very closely. The only exceptions are INS multilevel results with large η's. These, and also the corresponding results in Table 2, indicate greater sensitivity to large overweighting factors in the solution of the system.

Note that *only the momentum equation residuals need to be overweighted.* This is due to the fact that the continuity equation contributes an elliptic component (the Laplacian operator) to the system, for which overweighting is unnecessary. This point has been verified by two-level analysis of the linearized incompressible flow equations.

Experiments were also carried out with the advection-diffusion problem using two different η's. The average two-level convergence factor was 0.34 with the theoretically optimal two-level overweighting factors, and the average multilevel convergence factor was 0.43 when two prerelaxation sweeps and one postrelaxation sweep were employed. The large overweighting factors apparently cause problems. One of these is that overweighting undesirably amplifies high-frequency residuals as well as low-frequency ones, so more smoothing is necessary. Also, since the coarse-grid correction is improved, still more smoothing is required in order to reduce high-frequency errors sufficiently for the smaller convergence factor. Indeed, better performance was obtained when the number of relaxation sweeps per level was increased. With two prerelaxation and two postrelaxation sweeps, the convergence factor attained with the two-level cycle was 0.26 (rather than the theoretical 0.24), and the multilevel factor was 0.37 (rather than the theoretical 0.27). It seems that the assumptions of the analysis are no longer valid, since now intermediate eigenvalues also figure in the convergence rates, and these are neither real nor grid independent.

It is possible to find somewhat smaller overweighting factors that may provide improved results. This requires either a more sophisticated analysis that takes into account high-frequency phenomena, or numerical experimentation in the form of an automatic acceleration process. But in view of the results presented below, this does not seem to be the most profitable course.

5. The defect-correction W (DCW) cycle. The method of defect-correction iterations is a well-known tool for obtaining solutions with high-order accurate operators that are unstable, and that therefore cannot be used directly. This method can be embedded into the multigrid cycle as a means of accelerating convergence.

The basic defect-correction method is as follows. Suppose we wish to obtain an approximate solution to some equation

$$Lu = f \tag{36}$$

with suitable boundary conditions. We would like to use some finite difference operator L_2^h for approximating the differential operator L on a grid with mesh-size h, but cannot use it directly, say, due to problems of instability. Instead, we use another (usually lower-order) operator L_1^h, which is stable, and we hope to approach the L_2^h approximation via the following iterative process:

$$L_1^h u_i^h = f^h + (L_1^h - L_2^h) u_{i-1}^h\,, \tag{37}$$

where $i \geq 1$ is the iteration number and $u_0^h = 0$. Here again the h superscripts denote functions and operators defined on grid h. If this process converges, it must clearly converge to the solution with the operator L_2^h. In elliptic cases the convergence is usually fast for smooth solution components, in terms of which L_1^h is indeed a good approximation to L_2^h, while the slow convergence of nonsmooth components may actually be an advantage, since for them L_1^h may be better than L_2^h.

A form of this defect-correction process can be used for accelerating the convergence rate when it is slowed down by the poor coarse-grid correction to certain components. We present this technique here as employed in a W cycle, but the generalization to greater cycle indices γ is straightforward.

The two-level *correction scheme* DCW cycle for the solution of the discrete problem

$$L^h u^h = f^h \tag{38}$$

is defined as follows:

1. Start with some initial approximation to u^h on the fine grid h. Smooth the error corresponding to this approximation by some number of relaxation sweeps, obtaining $\tilde{u}^h$.

2. Transfer residuals to the coarse grid H, and solve the first coarse-grid problem

$$L^H v_1^H = r^H \equiv I_h^H \left(f^h - L^h \tilde{u}^h\right) , \tag{39}$$

where the H superscripts denote operators and functions defined on the coarse grid and I_h^H is some transfer operator from grid h to grid H.

3. Calculate and add to the right-hand side a defect-correction term, and solve the second coarse-grid problem:

$$L^H v_2^H = r^H + (L^H - L_h^H) v_1^H , \tag{40}$$

where L_h^H is some approximation to L^h on grid H, which is significantly better than L^H, as explained below.

4. Interpolate and add the correction v_2^H to $\tilde{u}^h$, and smooth this new approximation with some number of relaxation sweeps.

The operator L_h^H used in (40) is some higher-order grid H approximation to the fine-grid operator L^h, which is not used directly, e.g., because it cannot be smoothed efficiently. For example, if L^h and L^H are first-order upstream discretizations of the advection operator on grid h and H, respectively, then they can be viewed as second-order central difference approximations with added artificial viscosity that is proportional to the mesh-size. L_h^H can then be chosen to be a central-difference discretization plus artificial viscosity with coefficients that correspond to grid h, all discretized on grid H. This is then a second-order approximation to L^h. More generally, if L^h is an approximation of order q on grid h to the differential operator L, and, similarly, L^H is a qth-order approximation to L (and L^h) on grid H, then L_h^H can be any operator on grid H which is (at least) a $q+$ 1st-order approximation to L^h.

The step described in (40) is an attempt to improve the correction yielded by the coarse-grid operator. An alternative view comes to light when $L^H v_1^H$ is subtracted from both sides of (40), yielding

$$L^H (v_2^H - v_1^H) = r^H - L_h^H v_1^H . \tag{41}$$

Viewed thus, the DCW acts as two cycles, except that the visit to the fine grid between the cycles has been skipped, and the process of adding v_1^H to the initial fine-grid approximation and recalculating the residuals is approximated on the coarse grid instead. This viewpoint will later be useful in the analysis.

The multilevel DCW correction cycle is similarly defined, except that the coarse-grid problems are not solved exactly, but rather by a similar DCW cycle (each) on the coarser grid. This is done recursively, and only on the coarsest grid are the problems solved exactly. Note that the cycle index is two (W cycle), which is the reason for the name DCW.

5.1. DCW with FAS. The DCW cycle can of course be implemented with the full approximation scheme (FAS), but it is important to note that the defect correction process must be applied only to the *correction* given by the first "leg" of the W cycle, and not to the full solution. As in the usual FAS algorithm, (39) is now replaced by

$$L^H u_1^H = L^H \hat{I}_h^H \tilde{u}^h + r^H, \tag{42}$$

where $\hat{I}_h^H$ is some transfer operator from grid h to grid H, which need not be the same as I_h^H. Equation (40) is replaced by

$$L^H u_2^H = L^H \hat{I}_h^H \tilde{u}^h + r^H + (L^H - L_h^H)(u_1^H - \hat{I}_h^H \tilde{u}^h). \tag{43}$$

Finally, $u_2^H - \hat{I}_h^H \tilde{u}^h$ is interpolated and added to $\tilde{u}^h$ as in the usual FAS multigrid cycle. It is easy to verify that for a linear problem this process is equivalent to the correction-scheme two-level DCW cycle.

5.2. Two-level local mode analysis. We analyze the DCW cycle by the same infinite-space analysis employed in the previous sections, again assuming that the factor that determines the rate of convergence is the coarse-grid correction of smooth error components, whereas high-frequency error components are reduced sufficiently by relaxation. Also, intergrid transfers are again neglected, and the FDA is employed.

Suppose the current error in our approximation to the solution of (38) is

$$v^h = e^{i(\omega_1 x + \omega_2 y)}. \tag{44}$$

The corresponding first coarse-grid problem for the correction is then

$$L^H v_1^H = -L^h v^h. \tag{45}$$

Let $C = C(\omega_1, \omega_2) = \hat{L}^h(\omega_1, \omega_2)/\hat{L}^H(\omega_1, \omega_2)$, as in the previous sections. Now

$$v_1^H = -C v^h,$$

and the second coarse-grid problem, following (41), is

$$L^H (v_2^H - v_1^H) = -L^h v^h (1 - C), \tag{46}$$

where no distinction has been made between the actual fine-grid operator L^h and its coarse-grid approximation L_h^H, since the two are equivalent under the FDA. The solution to (46) is

$$v_2^H - v_1^H = -C(1 - C) v^h,$$

and the error-amplification factor ν after the correction has been added to the fine-grid solution is

$$\nu = \frac{v_{new}^h}{v^h} = \frac{v^h + v_2^H}{v^h} = (1 - C)^2. \tag{47}$$

This result is quite expected from the point of view of skipping the fine grid. That is, under the present simplifying assumptions the error-amplification factor per cycle is the same as that per two regular cycles as calculated in §2. In particular, the predicted two-level convergence factor per cycle for the first-order discretized advection-diffusion equation is 0.25.

5.3. Multilevel local mode analysis. Following the notation of §3 and the fine-grid skipping viewpoint, we find that, for the multilevel DCW cycle, the one-dimensional map describing the error-amplification factor ν^k (see derivation of (13) with $\gamma = 2$ for the regular W cycle) is given by

$$\nu^k = \left[1 - C(1 - \nu^{k-1})\right]^2 \tag{48}$$

for $k > 0$, with $\nu^0 = 0$. γ is omitted in the notation, since we are only considering a W cycle. Observe now that (48) could also have been obtained from (13) with $\gamma = 2$ by squaring both sides and substituting ν^k and ν^{k-1} for $(\nu_2^k)^2$ and $(\nu_2^{k-1})^2$. So the multilevel DCW cycle is seen to be equivalent to two regular W cycles under the assumptions of this analysis.

Conclusion. The convergence factor of the multilevel DCW cycle for the advection-diffusion equation with first-order discretization ($\xi = 0.5$) tends to one as the number of levels tends to infinity.

5.4. Combining the DCW cycle with OWR. We have seen that the DCW cycle itself does not solve the problem of poor convergence rates when the number of levels is very large, although the rate is squared for a given number of levels. A natural approach is to try to incorporate residual overweighting into the DCW cycle.

5.4.1. η-optimization. The proper way of applying residual overweighting in the DCW cycle is again best seen from the fine-grid skipping point of view. So considered, it is clear that not only do the residuals need to be multiplied by the overweighting factor η, but so does the term $-L_h^H v_1^H$ in (41), which approximates the term that would have been added to the fine-grid residual had the fine grid been visited. By (47), the resulting two-level error-amplification factor with single-parameter overweighting is given by

$$\nu_{tl} = (1 - \eta\xi)^2, \tag{49}$$

where ξ is again the minimal $\hat{L}^h/\hat{L}^H$.

The optimal η is of course the same as that for the regular cycle given in (21) and, by (22), ν_{tl} with the optimal two-level overweighting factor is given by

$$\nu_{tl} = \left(\frac{1-\xi}{1+\xi}\right)^2. \tag{50}$$

This yields a very satisfactory two-level convergence factor per cycle of 0.11 for the first-order discretized advection-diffusion equation.

The optimal multilevel overweighting factor η_{ml} is also the same as for the regular W cycle ($\xi^{-1/2}$), and the multilevel convergence factor, following (27)–(28), is

$$\nu_{ml} = (1 - \xi^{-1/2})^2. \tag{51}$$

For the first-order discretized advection-diffusion equation, $\xi = 0.5$ and $\nu_{ml} = 0.17$.

Multiparameter optimization is also performed as in the case of a regular cycle.

5.5. Defect-correcting for finer levels. In the DCW cycle as presented above, the defect corrections were all used to improve the approximation to the problem on the next-finer grid. But since it is only the *finest*-grid problem whose solution is sought, and not those of the intermediate grids, faster convergence may be obtained by employing operators of a much finer grid in the defect-correction stage. Clearly this will not affect two-level performance, since then the finest grid is used in the defect correction anyway, but multilevel performance may be improved.

Many schemes are possible, and we consider here the ultimate one of using the operator of the finest grid in all the defect corrections. Under the assumptions of the present analysis, in which the effect of relaxation is disregarded, this scheme is equivalent to employing *double discretization* (see [2]) within a W cycle. (In the double discretization method, *all* the residuals are calculated with the fine-grid operator, but since the initial solution on intermediate levels is zero and the effect of relaxation is disregarded, it does not matter which operator is used on the first leg of the W cycle.) Since now the scheme used on each grid depends on the finest mesh-size, or rather on the ratio of the current mesh-size to the finest one, the multilevel local mode analysis produces a family of recurrence equations rather than just one. Let $k = n - i$ again denote the difference between the finest level n and the solution level i, and let $\nu^{j,k}$ denote the error-amplification factor at level $i + j$ when the finest level is $n = i + k$. Suppose that the error in the approximation to the equation on level $i + j$ is v^j. Then the first leg of the DCW cycle produces a correction v_1^{j-1} of $-C(1 - \nu^{j-1,k})v^j$, as in the regular W cycle, but the second leg now produces an additional correction $v_2^{j-1} - v_1^{j-1}$ of $C(1 - \nu^{j-1,k})[-1 + C^{k-j+1}(1 - \nu^{j-1,k})]v^j$, rather than $C(1 - \nu^{j-1,k})[-1 + C(1 - \nu^{j-1,k})]v^j$ of the usual DCW cycle, since the ratio of the symbol of the operator on level $i + j - 1$ to that of the finest-level operator is C^{k-j+1}. Hence, $\nu^{j,k}$ is given by

$$\nu^{j,k} = 1 - 2C(1 - \nu^{j-1,k}) + C^{k-j+2}(1 - \nu^{j-1,k})^2 \,, \tag{52}$$

for $j \geq 1$ with $\nu^{0,k} = 0$.

From (52) it is clear that for large values of k some of the $\nu^{j,k}$'s may be quite large in absolute value. But these only express the error-amplification factors for intermediate grids, which are unimportant, because the purpose of the algorithm is to reduce the error on the *finest grid* as efficiently as possible. The only interesting value is therefore $\nu^{k,k}$, which is given by

$$\nu^{k,k} = (1 - C^k)^{2^k} \,. \tag{53}$$

This result can be proved from (52) by fixing k and performing an induction on j to show that the appropriate binomial expansion is obtained when $j = k$. The proof is omitted due to its length and its irrelevance to the main theme. However, the result is again clear from the fine-grid skipping point of view, although now *all* the levels that are finer than the solution level are skipped. The correction on the solution level is only C^k times the required correction, but 2^k visits are made to this level, in agreement with (53).

PROPOSITION 5. *The maximal $\nu^{k,k}$ over all $\xi \leq C \leq 1$ and $k \geq 1$ tends to e^{-1} for $\xi = 0.5$ and to 1 for $0 \leq \xi < 0.5$.*

Proof. For a fixed k and $0 \leq C \leq 1$, $\nu^{k,k}$ in (53) decreases monotonically as C increases. Therefore, the maximum is obtained when $C = \xi$. Also, for all $k \geq 1$ and $0 < \xi \leq 0.5$,

$$\begin{aligned}\frac{\nu^{k+1,k+1}}{\nu^{k,k}} &= \frac{(1-\xi^{k+1})^{2^{k+1}}}{(1-\xi^k)^{2^k}} \\ &= \left(\frac{(1-\xi\cdot\xi^k)^2}{1-\xi^k}\right)^{2^k} \\ &= \left(\frac{1-2\xi\cdot\xi^k+\xi^{2k+2}}{1-\xi^k}\right)^{2^k} = \left(1+\frac{(1-2\xi)\cdot\xi^k+\xi^{2k+2}}{1-\xi^k}\right)^{2^k} > 1.\end{aligned} \tag{54}$$

Hence, the maximal $\nu^{k,k}$ is obtained when $k \to \infty$, and is given by

$$\nu_{ml} = \lim_{k\to\infty}(1-\xi^k)^{2^k} = \lim_{k\to\infty} e^{-(2\xi)^k} = \begin{cases} 1, & 0<\xi<0.5, \\ e^{-1}, & \xi = 0.5. \end{cases} \tag{55}$$ □

Conclusion. The present DCW cycle yields an error-amplification factor of e^{-1} with a multilevel cycle for the first-order discretized advection-diffusion equation.

5.5.1. Combining defect corrections for the finest level with OWR. From (53) it may seem that the method of overweighted residuals cannot be profitably combined with defect corrections for the finest level since, if $C > 1$, $\nu^{k,k}$ diverges as $k \to \infty$. Suppose, however, that the residuals are all transferred with an overweighting factor of η, and the defect-correction term is also multiplied by this factor (rather than by η^{k-j+1}, as is implied by the fine-grid skipping viewpoint). This yields the following recurrence equation for the error-amplification factors $\nu^{j,k}$ (compare with (52)):

$$\nu^{j,k} = 1 - 2\eta C(1-\nu^{j-1,k}) + \eta^2 C^{k-j+2}(1-\nu^{j-1,k})^2 . \tag{56}$$

PROPOSITION 6. *For all* $0.5 \le C \le 1$ *and all* $k \ge 1$, (56) *with the optimal two-level overweighting factor* $\eta = 4/3$ *satisfies*

$$\nu^{k,k}(C) \le \nu^{1,1}(1) = \tfrac{1}{9} . \tag{57}$$

Rather than attempting the difficult task of proving Proposition 6 directly, let us note again that in the present algorithm the solution level can be viewed as solving directly for the correction for the finest grid, rather than for its next-finer grid. But it only yields C^k of the required correction per visit (multiplied by some coefficient that depends on the overweighting factor η). Therefore, $\nu^{k,k}$, which is a polynomial in C with coefficients that depend on η, can also be written as a polynomial in C^k. Let us define accordingly

$$\mu^{k,k}(C^k) \overset{\text{def}}{=} \nu^{k,k}(C) . \tag{58}$$

$\mu^{k,k}$ also describes the amplification factor of the fine-grid error, but in units of the solution-level correction, rather than the finest-level correction. Therefore, it satisfies a recurrence relationship similar to (56), but with C replaced by 1. In particular, $\mu^{j,k} = \mu^{j,j}$ for all $1 \le j \le k$, yielding the recurrence relationship

$$\mu^{k,k}(C^k) = \left[1-\eta(1-\mu^{k-1,k-1}(C^k))\right]^2 \tag{59}$$

for $k > 1$, with

$$\mu^{1,1}(C^k) = \left(1-\eta C^k\right)^2 . \tag{60}$$

From this relationship it is fairly straightforward to prove Proposition 6, since the conditions that $\mu^{k,k}$ must satisfy can be traced back to sufficient conditions on $\mu^{1,1}$, which can be shown by induction to be satisfied. We omit the proof since it is irrelevant to the main issues and somewhat lengthy.

Conclusion. Under the present assumptions the optimal two-level convergence factor of 0.11 is attainable with a multilevel cycle.

5.6. Numerical experiments. Numerical experiments were carried out with the same problems as for the method of OWR (§4.5). The fine mesh-size in the two-level experiments was again 1/64, and four levels were employed in the multilevel cycles, the finest mesh-size again being 1/128. The same discretizations were employed.

The operator L_h^H was chosen to be a central-difference approximation (to the advection operator), plus artificial viscosity with coefficients that approximate (to second order) the viscosity coefficients corresponding to grid h.

The results with the regular DCW cycle, as described in §5.4 with various overweighting factors η, appear in Table 2. Two prerelaxation and two postrelaxation sweeps were performed, and the experiments were repeated with three pre- and three postrelaxations (results in parentheses).

TABLE 2
Asymptotic residual convergence factors of numerical calculations for the AD *equation and* INS *equations, solved with a* DCW *cycle, are compared with analytical predictions for various single overweighting factors η. Two (three) pre- and two (three) postrelaxation sweeps were employed.*

η	Two-level			Multilevel		
	Analytical prediction	Numerical AD	Numerical INS	Analytical prediction	Numerical AD	Numerical INS
1.00	0.25	0.25	0.24 (0.24)	0.48	0.48	0.47 (0.46)
1.10	0.20	0.20	0.21 (0.21)	0.38	0.39	0.39 (0.39)
1.20	0.16	0.16	0.21 (0.20)	0.30	0.30	0.32 (0.33)
1.30	0.12	0.11	0.21 (0.19)	0.22	0.22	0.29 (0.27)
1.33	0.11	0.11	0.21 (0.16)	0.20	0.20	0.30 (0.26)
1.40	0.16	0.12	0.24 (0.20)	0.16	0.15	0.33 (0.29)
1.50	0.25	0.20	0.28 (0.25)	0.25	0.20	0.50 (0.50)

The numerical performance of the advection-diffusion solver again matches the prediction well. Performance obtained with large overweighting factors is somewhat better than predicted, especially when more relaxation sweeps are carried out. The reason for this is that, when η is larger than 4/3, the slowest components to converge have the second-finest level as their solution level. For these components, relaxation on the finest grid still has a nonnegligible effect, since their frequencies are fairly high. On the other hand, when η is smaller than 4/3, the slowest-converging components have solution levels that are very coarse, so that even if relaxation on their next-finer grid is very effective, the overall improvement in the convergence rate is very small. Results with the incompressible flow equations were not quite as good as predicted, although a very significant improvement was observed. These results improve further if a small amount of extra artificial viscosity is added, yielding a more effective smoother. When the viscosity coefficients are increased by 40 percent, for example, the asymptotic residual convergence factors with $\eta = 4/3$ and six relaxation sweeps per level improve to 0.14 (two-level) and 0.24 (multilevel). Of course, there is then a corresponding loss in accuracy, and the cost of performing more sweeps instead must be weighed against the cost of using slightly greater viscosity with a correspondingly finer grid to regain accuracy. Once again, as in the results listed in Table 1, greater overweighting factors than 4/3 yielded poorer results.

The multilevel experiments were repeated with the improvement of correcting the defect for the finest level, as introduced in §5.5. The results are compared with those predicted by numerical analysis of (56) in Table 3. Again, two (three) pre- and two (three) postrelaxations per level were performed. The numerical results match the predictions well for the advection-diffusion equation, although a total of four relaxation sweeps per level (with the present smoother) did not suffice to reduce the high-frequency errors enough. As more and more levels are used, the assumption of the analysis, that the only significant difference between the coarse-grid operators and the fine-grid ones is in the artificial viscosity, becomes poorer and poorer, since the correction level may be much coarser than the finest level. Hence, somewhat better results can be obtained by defect-correcting just a few levels up, and not all the way to the fine-grid operator. An optimal strategy may be worked out experimentally. The INS performance once again lags behind somewhat, but a highly significant improvement is shown, which is again increased by adding some artificial viscosity. When this viscosity is increased by 40 percent, the asymptotic residual convergence factor with $\eta = 4/3$ and six relaxation sweeps per level improves to 0.18. Once again, there is of course a loss of accuracy, which would require a correspondingly finer grid to offset. A slight further increase can be obtained by defect-correcting just a few levels up, rather than for the finest level.

TABLE 3

Asymptotic residual convergence factors of numerical calculations for the AD *equation and* INS *equations, solved with a* DCW *cycle with defect corrections, calculated by the finest-grid operator, are compared with analytical predictions for various single overweighting factors η, which are calculated numerically from* (56). *Two* (*three*) *pre- and two* (*three*) *postrelaxations were employed.*

η	Analytical prediction	Numerical AD	Numerical INS
1.00	0.37	0.36 (0.36)	0.36 (0.36)
1.10	0.23	0.23 (0.23)	0.28 (0.28)
1.20	0.16	0.16 (0.14)	0.26 (0.23)
1.30	0.12	0.15 (0.10)	0.29 (0.21)
1.33	0.11	0.15 (0.10)	0.31 (0.24)
1.40	0.16	0.20 (0.13)	0.36 (0.34)
1.50	0.25	0.31 (0.21)	0.80 (0.67)

6. Conclusions and remarks. Methods of acceleration of multigrid convergence have been developed, analyzed, and tested, the numerical results mostly matching the predictions of the analyses well. With the optimal method of combining residual overweighting and defect corrections within the W cycle, a multilevel convergence factor of about 0.2 has been obtained for the incompressible flow equations, and 0.1 for the advection-diffusion equations with first-order discretization. We note that large overweighting factors may show poorer behavior than expected for the Navier–Stokes solver, unless supplemented by extra relaxation sweeps. Some work may be saved by varying the number of sweeps per level.

Although the analyses and experiments were carried out for the case of vanishing diffusion coefficients, the methods apply to finite-viscosity calculations equally well. The optimal parameters are then reduced, and the results improve accordingly, but using the optimal parameters calculated herein will still yield convergence factors that are at least as good as in the vanishing-viscosity case. This has been verified experimentally, the convergence factors actually improving, even with the present optimal overweighting factors, apparently due to improved smoothing. Finally, these methods can also be used with anisotropic viscosity as in upstream differencing.

The methods as presented here are of quite limited value when used *directly* with second-order accurate discretization. Very large overweighting factors need then be applied, which also amplify high-frequency error components. An approach that appears promising is to employ overweighting in conjunction with upstream discretization and downstream ordering of relaxation. Early results with the advection-diffusion equation (with closed streamlines) have been successful, but only very simple examples have so far been attempted, and this approach requires extensive research.

Acknowledgment. We wish to thank Dr. Kees Venner, presently of the Weizmann Institute, for suggesting the particular averaging scheme used in the numerical experiments, and for verifying most of the numerical results reported with independent calculations.

REFERENCES

[1] D. Bai and A. Brandt, *Local mesh refinement multilevel techniques*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 109–134.

[2] A. Brandt, 1984 *Multigrid Guide with Applications to Fluid Dynamics*, Monograph, GMD-Studie 85, GMD-FIT, St. Augustin, West Germany, 1985.

[3] ———, *Multigrid Solvers for Non-Elliptic and Singular-Perturbation Steady-State Problems*, Report, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 1981.

[4] ———, *Rigorous local mode analysis of multigrid*, in Preliminary Proceedings, 4th Copper Mountain Conf. on Multigrid Methods, Chapter 2, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[5] A. Brandt and N. Dinar, *Multigrid solutions to elliptic flow problems*, in Numerical Methods for Partial Differential Equations 53, S. Parter, ed., Academic Press, 1979; ICASE Report 79–15, NASA Langley Research Center, Hampton, VA, 1979.

[6] A. Brandt and I. Yavneh, *Inadequacy of first-order upwind difference schemes for some recirculating flows*, J. Comput. Phys., 93 (1991), pp. 128–143.

[7] ———, *On multigrid solution of high-Reynolds incompressible entering flows*, J. Comput. Phys., 101 (1992), pp. 151–164.

[8] P. M. de Zeeuw and E. J. van Asselt, *The convergence rate of multi-level algorithms applied to the convection-diffusion equation*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 492–503.

[9] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.

[10] R. McLachlan, *Separated Viscous Flows via Multigrid*, Ph.D. thesis, California Institute of Technology, Pasadena, CA, 1990.

[11] N. N. Yanenko and Y. I. Shokin, *On the correctness of first differential approximation of difference schemes*, Dokl. Akad. Nauk. SSSR, 182 (1968), pp. 776–778.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 627–653, May 1993

SOLUTION OF ELLIPTIC SYSTEMS OF PARTIAL DIFFERENTIAL EQUATIONS BY CELL DISCRETIZATION*

JOHN GREENSTADT†

Abstract. The cell discretization algorithm is applied in a straightforward manner to a few very simple elliptic systems of partial differential equations. The discrete approximant of the solution depends, as usual, on sets of coefficients of intracell basis functions, one set for each cell. The usual transformation to new variables, one subset associated with interfaces and the other with cells, is easily generalized to the multivariable case. This renders the calculation almost entirely cellwise independent, and therefore readily parallelizable without special reorganization. The residual dependency in the calculation is in the solution of the system of equations for the interface variables. Because of the small sizes of all the equation systems for the uncoupled discrete variables, Gauss elimination was used throughout to solve them. Results for a simple gradient-divergence system, the Cauchy–Riemann equations, the Stokes problem, and the clamped-plate problem are shown in graphic form.

Key words. cell discretization, elliptic differential equation systems

AMS(MOS) subject classifications. 65N30, 65N35

1. Introduction.

In [4], the treatment of a nonselfadjoint elliptic partial differential equation (PDE) by the cell discretization (CD) method was described. We shall show here how that treatment may be generalized to handle systems of PDEs and show a few simple computed examples of the application of the resulting algorithm.

The standard equation of this type for a single function ψ of x in a domain Ω (in R^n) with boundary Γ is

$$-\nabla\cdot(\overleftrightarrow{a}\cdot\nabla\psi)+\vec{b}\cdot\nabla\psi+c\psi=d, \tag{1.1}$$

where all coefficients may depend on the coordinates and the double arrow over a indicates that it is a diadic (here assumed to be positive-definite symmetric). The boundary condition assumed to hold on Γ is

$$P(s)\,\psi(s)+Q(s)\,\frac{\partial\psi}{\partial n}(s)=R(s), \tag{1.2}$$

where the argument s indicates that the point which it labels is on Γ; points in Ω are denoted by x, y, etc. The left-hand side of (1.2) may be regarded as the result of a so-called *trace* operator $\mathcal{U}$ on the function $\psi(x)$ to produce a function of s defined on Γ.

We shall generalize this equation by first assuming that there are N unknowns $\{\psi_1, \psi_2, \ldots, \psi_N\}$, and that they satisfy the equations

$$\sum_{\beta=1}^{N}\left[-\nabla\cdot(\overleftrightarrow{a}_{\alpha\beta}\cdot\nabla\psi_\beta)+\vec{b}_{\alpha\beta}\cdot\nabla\psi_\beta+c_{\alpha\beta}\,\psi_\beta\right]=d_\alpha \tag{1.3}$$

for $\alpha=1, 2, \ldots, N$. This form includes most of the linear systems which are important in applications.

Part of the CD algorithm is the partitioning of the domain Ω, in which the equations are satisfied, into a set of disjoint (open) subdomains $\{\Omega_1, \Omega_2, \ldots, \Omega_K\}$

* Received by the editors May 29, 1989; accepted for publication (in revised form) May 11, 1992.

† Department of Applied Mathematics and Theoretical Physics, Silver Street, Cambridge CB3 9EW, England. Present address, 725 Mariposa Avenue, Mountain View, California 94041.

which, with their boundaries, together cover Ω. The boundary of Ω is denoted by Γ and the interface between contiguous cells, e.g., Ω_k and Ω_m, is denoted by Γ_{km}. It was pointed out in [3] that all of the interface conditions in all problems (when they are linearized) may be written in the form

$$\begin{aligned}P_{km}(s)\,\psi_k(s) + Q_{km}(s)\,\frac{\partial\psi_k}{\partial n}(s) - R_{km}(s)\\ = P_{mk}(s)\,\psi_m(s) + Q_{mk}(s)\,\frac{\partial\psi_m}{\partial n}(s) - R_{mk}(s),\end{aligned} \tag{1.4}$$

in which the various trace operators are defined with reference to their own cells only. This means that we may treat Ω_k as if it were in isolation, and the interface conditions on its bounding faces as if they were boundary conditions on ψ_k. We follow the plan, therefore, of generalizing (1.2) on the boundary of a single cell, and assume that

$$\sum_{\beta=1}^{N}\left[P_{c\beta}(s)\,\psi_\beta(s) + Q_{c\beta}(s)\,\frac{\partial\psi_\beta}{\partial n}(s)\right] = R_c(s) \tag{1.5}$$

with $c = 1, 2, \ldots, C$, the number of boundary conditions. Note that C need not be equal to N.

2. The variational process. Because of the first-order derivative terms in (1.3), it is convenient, as in [4], to introduce paired variables, primal and dual, instead of just one self-dual variable. We denote these by $\{\psi_{\mathbf{p}\alpha}(x)\}$ and $\{\psi_{\mathbf{d}\alpha}(x)\}$, respectively. A functional whose variation with respect to the $\psi_{\mathbf{d}\alpha}$ leads to (1.3) as the Euler equation is

$$\begin{aligned}\Phi_0 \equiv \sum_{\alpha\beta}\int_\Omega \Big\{\nabla\psi_{\mathbf{d}\alpha}\cdot\overleftrightarrow{a}_{\alpha\beta}\cdot\nabla\psi_{\mathbf{p}\beta} + \psi_{\mathbf{d}\alpha}\,\vec{\xi}_{\alpha\beta}\cdot\nabla\psi_{\mathbf{p}\beta} + (\vec{\eta}_{\alpha\beta}\cdot\nabla\psi_{\mathbf{d}\alpha})\,\psi_{\mathbf{p}\beta}\\ + (c_{\alpha\beta} + \nabla\cdot\vec{\eta}_{\alpha\beta})\,\psi_{\mathbf{d}\alpha}\,\psi_{\mathbf{p}\beta}\Big\}\,d\Omega\\ - \sum_{\alpha}\int_\Omega\Big\{\psi_{\mathbf{p}\alpha}\,d_{\mathbf{d}\alpha} + \psi_{\mathbf{d}\alpha}\,d_{\mathbf{p}\alpha}\Big\}\,d\Omega\end{aligned} \tag{2.1}$$

with $\vec{\xi} - \vec{\eta} = \vec{b}$ and with $d_{\mathbf{d}\alpha}$ undetermined. (For conciseness, we often suppress the primal-dual subscripts, where the context suffices to reduce ambiguity.)

As we shall see, it is feasible to carry through the entire treatment of the many-variable case in a much-simplified manner by grouping the set $\{\psi_{\mathbf{p}\alpha}\}$ into a single vector $\psi_{\mathbf{p}}$. The same is done with $\{\psi_{\mathbf{d}\alpha}\}$, and all other sets of variables depending on the indices α, β, γ, etc. We thus define

$$\begin{aligned}&\{\psi_{\mathbf{p}\alpha}\}\to\psi_{\mathbf{p}},\quad \{\psi_{\mathbf{d}\alpha}\}\to\psi_{\mathbf{d}},\quad \{\overleftrightarrow{a}_{\alpha\beta}\}\to\overleftrightarrow{a},\quad \{\vec{b}_{\alpha\beta}\}\to\vec{b},\\ &\{c_{\alpha\beta}\}\to c,\quad \{d_{\mathbf{p}\alpha}\}\to d_{\mathbf{p}},\quad \{d_{\mathbf{d}\alpha}\}\to d_{\mathbf{d}},\end{aligned} \tag{2.2}$$

which are all matrices or vectors in the *matrix* sense. However, $\overleftrightarrow{a}$ and $\vec{b}$ are also a diadic (matrix) and a vector, respectively, in the *geometrical* sense. These must be kept conceptually distinct; hence, when we write, e.g., $\vec{b}^T$, the transpose refers to the matrix aspect only. With this understanding, (2.1) may be condensed to

$$\begin{aligned}\Phi_0 \equiv \int_\Omega \Big\{ &\nabla\psi_{\mathbf{d}}^T \cdot \overleftrightarrow{a} \cdot \nabla\psi_{\mathbf{p}} + \psi_{\mathbf{d}}^T\, \vec{\xi}\cdot\nabla\psi_{\mathbf{p}} + (\nabla\psi_{\mathbf{d}}^T\cdot\vec{\eta})\,\psi_{\mathbf{p}} \\ &+\psi_{\mathbf{d}}^T\,(c+\nabla\cdot\vec{\eta})\,\psi_{\mathbf{p}}\Big\}\,d\Omega \\ -\int_\Omega &\Big\{\psi_{\mathbf{p}}^T\,d_{\mathbf{d}} + \psi_{\mathbf{d}}^T\,d_{\mathbf{p}}\Big\}\,d\Omega,\end{aligned} \tag{2.3}$$

which we integrate by parts (Green's formula) to obtain a more convenient form to be used later:

$$\begin{aligned}\Phi_0 = &\int_\Omega \psi_{\mathbf{d}}^T\Big\{-\nabla\cdot(\overleftrightarrow{a}\cdot\nabla\psi_{\mathbf{p}}) + (\vec{\xi}-\vec{\eta})\cdot\nabla\psi_{\mathbf{p}} + c\,\psi_{\mathbf{p}} - d_{\mathbf{p}}\Big\}\,d\Omega \\ &+\int_\Gamma \psi_{\mathbf{d}}^T\,\{\hat{n}\cdot\overleftrightarrow{a}\cdot\nabla\psi_{\mathbf{p}} + \hat{n}\cdot\vec{\eta}\,\psi_{\mathbf{p}}\}\,d\Gamma - \int_\Omega \psi_{\mathbf{p}}^T d_{\mathbf{d}}\,d\Omega.\end{aligned} \tag{2.4}$$

To condense the notation for (1.5), we first define the trace operator $\mathcal{U}_c\psi$ to mean

$$\begin{aligned}\mathcal{U}_c\psi &\equiv \mathcal{U}_c[\psi_1,\psi_2,\ldots,\psi_N] \equiv \sum_{\alpha=1}^{N}\,[\mathcal{U}_{c\alpha}\psi_\alpha]\,(s) \\ &\equiv \sum_{\alpha=1}^{N}\left[P_{c\alpha}(s)\,\psi_\alpha(s) + Q_{c\alpha}(s)\,\frac{\partial\psi_\alpha}{\partial n}(s)\right],\end{aligned} \tag{2.5}$$

and the whole collection can be written

$$\mathcal{U}\psi = R, \tag{2.6}$$

with

$$\mathcal{U}\psi = \begin{bmatrix}\mathcal{U}_1\psi \\ \mathcal{U}_2\psi \\ \vdots \\ \mathcal{U}_C\psi\end{bmatrix} \tag{2.7a}$$

and

$$R = \begin{bmatrix}R_1 \\ R_2 \\ \vdots \\ R_C\end{bmatrix}. \tag{2.7b}$$

Note that $\mathcal{U}\psi$ may also be written

$$\mathcal{U}\psi \equiv P(s)\psi(s) + Q(s)\frac{\partial\psi}{\partial n}(s), \tag{2.8}$$

where P and Q are $(C \times N)$ matrix functions of s.

If we define the vector Lagrange multiplier

$$\lambda_{\mathbf{p}}(s) \equiv \begin{bmatrix}\lambda_{\mathbf{p}1}(s) \\ \lambda_{\mathbf{p}2}(s) \\ \vdots \\ \lambda_{\mathbf{p}C}(s)\end{bmatrix} \tag{2.9}$$

with a similar definition for $\lambda_{\mathrm{d}}(s)$, the augmented functional Φ which incorporates the boundary conditions becomes

$$\Phi \equiv \Phi_0 + \int_\Gamma \lambda_{\mathrm{d}}^T \{\mathcal{U}_{\mathrm{p}}\psi_{\mathrm{p}} - R_{\mathrm{p}}\}\, d\Gamma + \int_\Gamma \lambda_{\mathrm{p}}^T \{\mathcal{U}_{\mathrm{d}}\psi_{\mathrm{d}} - R_{\mathrm{d}}\}\, d\Gamma. \tag{2.10}$$

To derive the constrained primal Euler equation, we vary ψ_{d}, denoting its variation by $\delta\psi_{\mathrm{d}}$. The result is (cf., for example, [1])

$$\begin{aligned}\delta\Phi = &\int_\Omega \delta\psi_{\mathrm{d}}^T\Big\{-\nabla\cdot\overset{\leftrightarrow}{a}\cdot\nabla\psi_{\mathrm{p}} + (\vec{\xi}-\vec{\eta})\cdot\nabla\psi_{\mathrm{p}} + c\psi_{\mathrm{p}} - d_{\mathrm{p}}\Big\}\, d\Omega\\ &+ \int_\Gamma \delta\psi_{\mathrm{d}}^T\{\hat{n}\cdot\overset{\leftrightarrow}{a}\cdot\nabla\psi_{\mathrm{p}} + \hat{n}\cdot\vec{\eta}\psi_{\mathrm{p}}\}\, d\Gamma\\ &+ \int_\Gamma \left\{\delta\psi_{\mathrm{d}}^T P_{\mathrm{d}}^T + \left(\delta\frac{\partial\psi_{\mathrm{d}}}{\partial n}\right)^T Q_{\mathrm{d}}^T\right\}\lambda_{\mathrm{p}}\, d\Gamma,\end{aligned} \tag{2.11}$$

which must vanish for a stationary Φ. Note that we have taken the matrix transpose of the last integrand for convenience later on.

The term $\hat{n}\cdot\overset{\leftrightarrow}{a}\cdot\nabla\psi$ is often called the *coderivative* of ψ and, for the Laplace operator, is the standard normal derivative. We shall denote it by $\partial\psi/\partial n_a$. The expression $\hat{n}\cdot\vec{\eta}$, we shall denote by η_n. Note that $\partial\psi/\partial n_a$ is a *linear combination* of the gradients of several (possibly all) of the functions $\{\psi_\alpha\}$. The relationship is

$$\frac{\partial\psi_\alpha}{\partial n_a} \equiv \sum_\beta \hat{n}\cdot\overset{\leftrightarrow}{a}_{\alpha\beta}\cdot\nabla\psi_\beta. \tag{2.12}$$

3. Analysis of boundary conditions. We start by introducing new variables to replace ψ and its normal derivative. We have made a necessary distinction between the coderivative and the ordinary normal derivative (or, in any case, whatever kind of normal derivative is defined for use in the imposed boundary conditions). Let us use the more compact notation of having ψ_a mean the coderivative based on the diadic $\overset{\leftrightarrow}{a}$, and of having ψ_n mean the imposed normal derivative. We wish to replace $\psi(s)$ and $\psi_n(s)$ by new variables $\sigma(s)$ and $\tau(s)$ by means of a linear transformation.

This transformation is based on the matrices $P(s)$ and $Q(s)$ which we first arrange as a single composite matrix $[P, Q]$ with C rows and $2N$ columns. We assume that this matrix has full row rank, ensuring that the boundary conditions are linearly independent. It is then possible, by column combinations (i.e., elementary transformations from the right) to reduce $[P, Q]$ to a simple form. If the product of all the transformations is denoted by Y—a $(2N \times 2N)$ nonsingular matrix—then we have

$$[P, Q] \times Y = [I, 0], \tag{3.1}$$

where I is a unit matrix of order $(C \times C)$. The matrix Y can be partitioned as follows:

$$Y = \begin{bmatrix} J & K \\ L & M \end{bmatrix}, \tag{3.2}$$

and Y can in turn be inverted to yield back $[P, Q]$ as well as the submatrices R and S. The complete matrices then satisfy

$$\begin{bmatrix} P & Q \\ R & S \end{bmatrix} \times \begin{bmatrix} J & K \\ L & M \end{bmatrix} = \begin{bmatrix} PJ + QL, & PK + QM \\ RJ + SL, & RK + SM \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{3.3}$$

It is important to note that all these relations hold, by construction, for all s on Γ. Also, if $2N - C \equiv D$, then the orders of the various matrices are as follows: P, Q are $(C \times 2N)$; R, S are $(D \times 2N)$; J, K are $(2N \times C)$; and L, M are $(2N \times D)$.

The new variables are introduced by

$$\begin{bmatrix} \sigma \\ \tau \end{bmatrix} = \begin{bmatrix} P & Q \\ R & S \end{bmatrix} \begin{bmatrix} \psi \\ \psi_n \end{bmatrix} \tag{3.4}$$

so that σ is a vector function of length C and τ is a vector function of length D, while ψ and ψ_n are both of length N. The inverse of (3.4) is

$$\begin{bmatrix} \psi \\ \psi_n \end{bmatrix} = \begin{bmatrix} J & K \\ L & M \end{bmatrix} \begin{bmatrix} \sigma \\ \tau \end{bmatrix}. \tag{3.5}$$

To proceed further with the analysis of the boundary conditions, we assume that the Euler equation holds in the interior, so that the integral over Ω in (2.11) may be ignored. Since we shall be varying the dual quantities, we can also ignore the other volume integral. The remaining terms relate to the surface only, so we shall denote what remains of the functional by Φ_S, which is

$$\begin{aligned} \Phi_S = &\int_\Gamma \psi_{\mathrm{d}}^T \{\psi_{\mathrm{p}a} + \eta b_n \psi_{\mathrm{p}}\}\, d\Gamma + \int_\Gamma \lambda_{\mathrm{d}}^T \{P_{\mathrm{p}}\psi_{\mathrm{p}} + Q_{\mathrm{p}}\psi_{\mathrm{p}n} - R_{\mathrm{p}}\}\, d\Gamma \\ &+ \int_\Gamma \lambda_{\mathrm{p}}^T \{P_{\mathrm{d}}\psi_{\mathrm{d}} + Q_{\mathrm{d}}\psi_{\mathrm{d}n} - R_{\mathrm{d}}\}\, d\Gamma. \end{aligned} \tag{3.6}$$

Here we treat the simplest case when $\psi_{\mathrm{p}n} = \psi_{\mathrm{p}a}$, and substitute for ψ and ψ_a in terms of σ and τ. The result is

$$\begin{aligned} \Phi_S = &\int_\Gamma \left(\sigma_{\mathrm{d}}^T J_{\mathrm{d}}^T + \tau_{\mathrm{d}}^T K_{\mathrm{d}}^T\right) \left[L_{\mathrm{p}}\sigma_{\mathrm{p}} + M_{\mathrm{p}}\tau_{\mathrm{p}} + \eta b_n (J_{\mathrm{p}}\sigma_{\mathrm{p}} + K_{\mathrm{p}}\tau_{\mathrm{p}})\right] d\Gamma \\ &+ \int_\Gamma \lambda_{\mathrm{d}}^T \{\sigma_{\mathrm{p}} - R_{\mathrm{p}}\}\, d\Gamma + \int_\Gamma \lambda_{\mathrm{p}}^T \{\sigma_{\mathrm{d}} - R_{\mathrm{d}}\}\, d\Gamma. \end{aligned} \tag{3.7}$$

The result of varying σ_{d} is just a formula for λ_{p}, but varying τ_{d} yields a consistency condition, which is

$$\begin{aligned} &K_{\mathrm{d}}^T \left[L_{\mathrm{p}}\sigma_{\mathrm{p}} + M_{\mathrm{p}}\tau_{\mathrm{p}} + \eta b_n (J_{\mathrm{p}}\sigma_{\mathrm{p}} + K_{\mathrm{p}}\tau_{\mathrm{p}})\right] \\ &\quad = K_{\mathrm{d}}^T (L_{\mathrm{p}} + \eta b_n J_{\mathrm{p}})\sigma_{\mathrm{p}} + K_{\mathrm{d}}^T (M_{\mathrm{p}} + \eta b_n K_{\mathrm{p}})\tau_{\mathrm{p}} \\ &\quad \equiv A_{\mathrm{p}}\sigma_{\mathrm{p}} + B_{\mathrm{p}}\tau_{\mathrm{p}} = 0. \end{aligned} \tag{3.8}$$

Since this is an identity in σ and τ, the two terms must vanish individually. This means, first, that

$$B_{\mathrm{p}} = 0. \tag{3.9}$$

σ_{p} may be evaluated by varying λ_{d} in (3.7). The result is

$$\sigma_{\mathrm{p}} = R_{\mathrm{p}}. \tag{3.10}$$

Hence, it is only necessary to have

$$A_{\mathrm{p}} R_{\mathrm{p}} = 0. \tag{3.11}$$

Conditions (3.9) and (3.11) constitute a concise test as to whether the "nullification" of a conflicting induced boundary condition by means of a boundary correction is necessary.

Appropriate modifications must be made in case $\psi_n \neq \psi_a$.

4. Construction of the nullifier. We shall follow the treatment in [5], but we shall keep track of the matrix relationships among the quantities arising in the many-variable case. We first adjoin to Φ_S another surface term Φ_N

$$\Phi^* \equiv \Phi_S - \Phi_N, \tag{4.1}$$

where

$$\Phi_N \equiv \int_\Gamma N(\sigma_{\mathbf{p}}, \tau_{\mathbf{p}}, \sigma_{\mathbf{d}}, \tau_{\mathbf{d}})\, d\Gamma. \tag{4.2}$$

The arguments of N (the "nullifier") are all vector variables.

When $\tau_{\mathbf{d}}$ is varied, the effect on Φ^* is

$$\delta\Phi^* = \int_\Gamma \delta\tau_{\mathbf{d}}(A\sigma_{\mathbf{p}} + B\tau_{\mathbf{p}})\, d\Gamma - \int_\Gamma \delta\tau_{\mathbf{d}} \frac{\partial N}{\partial \tau_{\mathbf{d}}}\, d\Gamma. \tag{4.3}$$

Hence, for a stationary Φ^*, we must have

$$\frac{\partial N}{\partial \tau_{\mathbf{d}}} = A\sigma_{\mathbf{p}} + B\tau_{\mathbf{p}}. \tag{4.4}$$

Similarly, it not hard to see that $\partial N/\partial \rho_{\mathbf{d}} = 0$. Hence, it is clear that

$$N = {\tau_{\mathbf{d}}}^T(A\sigma_{\mathbf{p}} + B\tau_{\mathbf{p}}) + \chi(\sigma_{\mathbf{p}}, \tau_{\mathbf{p}}, \sigma_{\mathbf{d}}), \tag{4.5}$$

where χ is an arbitrary function of its arguments. A particular choice of χ will serve to symmetrize N in some cases. This choice yields

$$N = {\tau_{\mathbf{d}}}^T A\sigma_{\mathbf{p}} + {\sigma_{\mathbf{d}}}^T A^T \tau_{\mathbf{p}} + {\tau_{\mathbf{d}}}^T B\tau_{\mathbf{p}}. \tag{4.6}$$

For a selfadjoint problem with $\vec{g} = \vec{a}$, we have $\sigma_{\mathbf{d}} = \sigma_{\mathbf{p}}$ and $\tau_{\mathbf{d}} = \tau_{\mathbf{p}}$, so interchanging the primal and dual variables has no effect. N is therefore symmetric.

Finally, we wish to restore the original variables ψ and ψ_g. This can be done most compactly by using the trace operator $\mathcal{U}$ and two new ones: $\mathcal{X}$ and $\mathcal{Y}$. These are defined as follows:

$$\sigma = P\psi + Q\psi_g \equiv \mathcal{U}\psi, \qquad \tau = R\psi + S\psi_g \equiv \mathcal{X}\psi, \tag{4.7}$$

so that N becomes

$$N = (\mathcal{X}_{\mathbf{d}}\psi_{\mathbf{d}})^T A(\mathcal{U}_{\mathbf{p}}\psi_{\mathbf{p}}) + (\mathcal{U}_{\mathbf{d}}\psi_{\mathbf{d}})^T A^T(\mathcal{X}_{\mathbf{p}}\psi_{\mathbf{p}}) + (\mathcal{X}_{\mathbf{d}}\psi_{\mathbf{d}})^T B(\mathcal{X}_{\mathbf{p}}\psi_{\mathbf{p}}). \tag{4.8}$$

The nullifier is rather more complicated when $\psi_n \neq \psi_a$.

5. Discretization. In the CD method, the boundary (and interface) conditions are imposed by the *moment collocation* process. For a single variable ψ and a single boundary condition, a set of *weight functions* $\{w_\kappa(s); \kappa = 1, \ldots, L\}$ is chosen. They are linearly independent and are capable of approximating any function in an appropriate class if L is large enough. (A system like this is called a Schauder basis.) For a single variable, we impose the collocation by forcing L moments of the boundary condition to vanish:

$$\int_\Gamma w_\kappa(s)\{\mathcal{U}\psi - R\}(s)\, d\Gamma = 0, \qquad \kappa = 1, \ldots, L. \tag{5.1a}$$

With primal and dual labels, (5.1a) should read

$$\begin{aligned}\int_\Gamma w_{\mathbf{d}\kappa}(s)\,\{\mathcal{U}_{\mathbf{p}}\psi_{\mathbf{p}} - R_{\mathbf{p}}\}\,(s)\,d\Gamma &= 0, \qquad \kappa = 1,\dots,L,\\ \int_\Gamma w_{\mathbf{p}\kappa}(s)\,\{\mathcal{U}_{\mathbf{d}}\psi_{\mathbf{d}} - R_{\mathbf{d}}\}\,(s)\,d\Gamma &= 0, \qquad \kappa = 1,\dots,L.\end{aligned} \tag{5.1b}$$

When there are C boundary conditions, and $C > 1$, we need correspondingly more sets of weight functions, one set for each boundary condition; e.g., $\{w_{c\kappa}(s); \kappa = 1,\dots,L(c); c = 1,\dots,C\}$ with the corresponding discrete boundary conditions

$$\int_\Gamma w_{c\kappa}\,\{\mathcal{U}\psi - R\}\;d\Gamma = 0; \quad \kappa = 1,\dots,L(c); \quad c = 1,\dots,C. \tag{5.2}$$

Combining this with (2.5), we obtain

$$\int_\Gamma w_{c\kappa}(s)\left\{\sum_{\alpha=1}^{N}\,[\mathcal{U}_{c\alpha}\psi_\alpha]\,(s) - R_c(s)\right\}\,d\Gamma = 0. \tag{5.3}$$

Following the usual procedure, we would now substitute a linear combination of basis functions for ψ. However, since we have N distinct ψ's, we may define N separate basis sets $\{\phi_{\alpha\mu}(x); \mu = 1,\dots,M(\alpha); \alpha = 1,\dots,N\}$ and set

$$\psi_\alpha(x) = \sum_{\mu=1}^{M(\alpha)} \phi_{\alpha\mu}(x)\theta_{\alpha\mu}. \tag{5.4}$$

Substituting this into (5.3) yields

$$\int_\Gamma w_{c\kappa}(s)\left\{\sum_{\alpha=1}^{N} \mathcal{U}_{c\alpha} \sum_{\mu=1}^{M(\alpha)} \phi_{\alpha\mu}(x)\theta_{\alpha\mu} - R_c(s)\right\}\,d\Gamma = 0. \tag{5.5}$$

To simplify the equations somewhat, we shall not always indicate the dependence of L on c and of M on α. We shall also not exhibit the arguments x and s, but leave their determination to the context. We then have

$$\begin{aligned}&\int_\Gamma w_{c\kappa}\left\{\sum_\alpha \mathcal{U}_{c\alpha}\sum_\mu \phi_{\alpha\mu}\theta_{\alpha\mu} - R_c\right\}\,d\Gamma\\ &= \int_\Gamma w_{c\kappa}\left\{\sum_\alpha\sum_\mu \mathcal{U}_{c\alpha}\phi_{\alpha\mu}\theta_{\alpha\mu} - R_c\right\}\,d\Gamma = 0\\ &= \sum_\alpha\sum_\mu \int_\Gamma [w_{c\kappa}\mathcal{U}_{c\alpha}\phi_{\alpha\mu}]\,d\Gamma\;\theta_{\alpha\mu} - \int_\Gamma w_{c\kappa}R_c\,d\Gamma = 0.\end{aligned} \tag{5.6}$$

If we define the arrays

$$\begin{aligned}U_{c\kappa,\alpha\mu} &\equiv \int_\Gamma w_{c\kappa}(s)\,[\mathcal{U}_{c\alpha}\phi_{\alpha\mu}]\,(s)d\Gamma(s),\\ R_{c\kappa} &\equiv \int_\Gamma w_{c\kappa}(s)R_c(s)\,d\Gamma(s),\end{aligned} \tag{5.7}$$

then (5.6) takes the simple form

$$\sum_{\alpha\mu} U_{c\kappa,\alpha\mu}\theta_{\alpha\mu} - R_{c\kappa} = 0. \tag{5.8}$$

The next simplification in notation is to combine c and κ into one "global" index, κ^*. To do this, we enumerate c, κ in the sequence

$$\begin{aligned}\{c,\kappa\} = \{&(1,1),(1,2),\dots,(1,L(1)),(2,1),(2,2),\dots,\\ &(2,L(2)),\dots,(C,1),(C,2),\dots,(C,L(C))\},\end{aligned} \tag{5.9}$$

while

$$\{\kappa^*\} = \{1,2,3,\dots,L^*\}.$$

This equation is an attempt to show that the composite index κ^* runs directly from 1 to L^*, the global collocation count, while the pair (c,κ) runs in the sequence shown. In like manner, we collect (α,μ) into one index μ^*, which runs from 1 to M^*, the global basis index count. In this way, we can condense (5.8) to

$$\sum_{\mu^*=1}^{M^*} U_{\kappa^*\mu^*}\theta_{\mu^*} - R_{\kappa^*} = 0, \tag{5.10}$$

which, in turn, can be written in terms of global matrices and vectors:

$$U\theta - R = 0, \tag{5.11}$$

where U is an $(L^* \times M^*)$ matrix, and θ and R are vectors of, respectively, M^* and L^* elements.

Next, we return to Φ_0, and substitute the expression (5.4) for ψ into (2.1). We get

$$\Phi_0 \equiv \sum_{\alpha\mu\beta\nu} S_{\alpha\mu;\beta\nu}\theta_{\mathbf{d}\alpha\mu}\theta_{\mathbf{p}\beta\nu} - \sum_{\alpha\mu}\theta_{\mathbf{p}\alpha\mu}T_{\mathbf{d}\alpha\mu} - \sum_{\alpha\mu}\theta_{\mathbf{d}\alpha\mu}T_{\mathbf{p}\alpha\mu}, \tag{5.12}$$

where

$$\begin{aligned} S_{\alpha\mu;\beta\nu} \equiv \int_\Omega \Big\{ &\nabla\phi_{\mathbf{d}\alpha\mu}\cdot\overleftrightarrow{a}_{\alpha\beta}\cdot\nabla\phi_{\mathbf{p}\beta\nu} + \phi_{\mathbf{d}\alpha\mu}\,\vec{\xi}_{\alpha\beta}\cdot\nabla\phi_{\mathbf{p}\beta\nu} + (\vec{\eta}_{\alpha\beta}\cdot\nabla\phi_{\mathbf{d}\alpha\mu})\,\phi_{\mathbf{p}\beta\nu} \\ &+ (c_{\alpha\beta} + \nabla\cdot\vec{\eta}_{\alpha\beta})\,\phi_{\mathbf{d}\alpha\mu}\,\phi_{\mathbf{p}\beta\nu}\Big\}\,d\Omega, \\ T_{\mathbf{d}\alpha\mu} &\equiv \int_\Omega \phi_{\mathbf{p}\alpha\mu}\,d_{\mathbf{d}\alpha}\,d\Omega, \\ T_{\mathbf{p}\alpha\mu} &\equiv \int_\Omega \phi_{\mathbf{d}\alpha\mu}\,d_{\mathbf{p}\alpha}\,d\Omega. \end{aligned} \tag{5.13}$$

By combining α and μ into μ^* and β and ν into ν^*, (5.12) becomes

$$\Phi_0 \equiv \sum_{\mu^*\nu^*} S_{\mu^*\nu^*}\theta_{\mathbf{d}\mu^*}\theta_{\mathbf{p}\nu^*} - \sum_{\mu^*}\theta_{\mathbf{p}\mu^*}T_{\mathbf{d}\mu^*} - \sum_{\mu^*}\theta_{\mathbf{d}\mu^*}T_{\mathbf{p}\mu^*}. \tag{5.14}$$

With this system of labeling, S and T play the same roles for several variables as do their counterparts in the single-variable case [4]. Therefore, we can go through all the standard procedures, always understanding that the modified labeling is in force. In particular, we can write

$$\Phi_0 \equiv \theta_{\mathbf{d}}^T S\theta_{\mathbf{p}} - \theta_{\mathbf{p}}^T T_{\mathbf{d}} - \theta_{\mathbf{d}}^T T_{\mathbf{p}}. \tag{5.15}$$

The completed functional, incorporating the boundary conditions is

$$\Phi \equiv \theta_{\mathbf{d}}^T S\theta_{\mathbf{p}} - \theta_{\mathbf{p}}^T T_{\mathbf{d}} - \theta_{\mathbf{d}}^T T_{\mathbf{p}} + \lambda_{\mathbf{d}}^T \left[U_{\mathbf{p}}\theta_{\mathbf{p}} - R_{\mathbf{p}}\right] + \lambda_{\mathbf{p}}^T \left[U_{\mathbf{d}}\theta_{\mathbf{d}} - R_{\mathbf{d}}\right], \tag{5.16}$$

and the discrete equations are

$$\begin{aligned} \frac{\partial \Phi}{\partial \theta_{\mathbf{d}}} &= S\theta_{\mathbf{p}} - T_{\mathbf{p}} + U_{\mathbf{d}}^T \lambda_{\mathbf{p}} = 0, \\ \frac{\partial \Phi}{\partial \theta_{\mathbf{p}}} &= S\theta_{\mathbf{d}} - T_{\mathbf{d}} + U_{\mathbf{p}}^T \lambda_{\mathbf{d}} = 0. \end{aligned} \tag{5.17}$$

We note again that the orders of the various quantities are $S{:}(M^* \times M^*)$; $\theta{:}(M^* \times 1)$; $T{:}(L^* \times 1)$; $U{:}(L^* \times M^*)$; $\lambda{:}(L^* \times 1)$.

We next construct the auxiliary matrices $V_{\mathbf{p}}$, $V_{\mathbf{d}}$, $Z_{\mathbf{p}}$, and $Z_{\mathbf{d}}$ of the following orders: $V_{\mathbf{p}}$ and $V_{\mathbf{d}}$ are $(M^* \times L^*)$ and $Z_{\mathbf{p}}$ and $Z_{\mathbf{d}}$ are $(M^* \times J^*)$, where $J^* \equiv M^* - L^*$. They have the properties

$$\begin{aligned} U_{\mathbf{p}}V_{\mathbf{p}} &= U_{\mathbf{d}}V_{\mathbf{d}} = I, \\ U_{\mathbf{p}}Z_{\mathbf{p}} &= U_{\mathbf{d}}Z_{\mathbf{d}} = 0, \\ Z_{\mathbf{d}}^T S V_{\mathbf{p}} &= V_{\mathbf{d}}^T S Z_{\mathbf{p}} = 0. \end{aligned} \tag{5.18}$$

The dominant terms in the number of standard floating-point operations needed to compute $V_{\mathbf{p}}$ and $Z_{\mathbf{p}}$ *for each cell* are $3M^{*3} + L^{*2}M^* - 2L^*M^{*2}$. We emphasize again that these calculations for a given cell are independent of those for any other cell. We use these matrices to transform from $\theta_{\mathbf{p}}$ and $\theta_{\mathbf{d}}$ to new variables $\sigma_{\mathbf{p}}$, $\sigma_{\mathbf{d}}$, $\rho_{\mathbf{p}}$, and $\rho_{\mathbf{d}}$ as follows:

$$\theta_{\mathbf{p}} = V_{\mathbf{p}}\sigma_{\mathbf{p}} + Z_{\mathbf{p}}\rho_{\mathbf{p}}, \qquad \theta_{\mathbf{d}} = V_{\mathbf{d}}\sigma_{\mathbf{d}} + Z_{\mathbf{d}}\rho_{\mathbf{d}}. \tag{5.19}$$

The σ's are $(L^* \times 1)$ and the ρ's are $(J^* \times 1)$. This transformation serves two important purposes: (1) It simplifies the boundary condition so drastically that it can almost be eliminated; (2) It decouples the σ's and ρ's so that they can be solved for completely independently. The latter is very important, in that it makes almost all of the calculations independent from cell to cell (as we shall see later) and hence makes the calculations almost completely parallelizable without any extra manipulation.

To show the first effect, we substitute for θ in (5.11) (both primal and dual variables satisfy the same relation):

$$\begin{aligned} U\theta - R &= U(V\sigma + Z\rho) - R = UV\sigma + UZ\rho - R \\ &= I \cdot \sigma + 0 \cdot \rho - R = \sigma - R = 0, \end{aligned} \tag{5.20}$$

which yields a trivially simple solution for σ.

Next, we substitute from (5.19) into (5.16) with the result

$$
\begin{aligned}
\Phi &\equiv [V_{\mathbf{d}}\sigma_{\mathbf{d}} + Z_{\mathbf{d}}\rho_{\mathbf{d}}]^T S\,[V_{\mathbf{p}}\sigma_{\mathbf{p}} + Z_{\mathbf{p}}\rho_{\mathbf{p}}] \\
&\quad - [V_{\mathbf{p}}\sigma_{\mathbf{p}} + Z_{\mathbf{p}}\rho_{\mathbf{p}}]^T T_{\mathbf{d}} - [V_{\mathbf{d}}\sigma_{\mathbf{d}} + Z_{\mathbf{d}}\rho_{\mathbf{d}}]^T T_{\mathbf{p}} \\
&\quad + \lambda_{\mathbf{d}}^T[\sigma_{\mathbf{p}} - R_{\mathbf{p}}] + \lambda_{\mathbf{p}}^T[\sigma_{\mathbf{d}} - R_{\mathbf{d}}] \\
&= \sigma_{\mathbf{d}}^T V_{\mathbf{d}}^T S V_{\mathbf{p}}\sigma_{\mathbf{p}} + \sigma_{\mathbf{d}}^T V_{\mathbf{d}}^T S Z_{\mathbf{p}}\rho_{\mathbf{p}} + \rho_{\mathbf{d}}^T Z_{\mathbf{d}}^T S V_{\mathbf{p}}\sigma_{\mathbf{p}} + \rho_{\mathbf{d}}^T Z_{\mathbf{d}}^T S Z_{\mathbf{p}}\rho_{\mathbf{p}} \\
&\quad - [V_{\mathbf{p}}\sigma_{\mathbf{p}} + Z_{\mathbf{p}}\rho_{\mathbf{p}}]^T T_{\mathbf{d}} - [V_{\mathbf{d}}\sigma_{\mathbf{d}} + Z_{\mathbf{d}}\rho_{\mathbf{d}}]^T T_{\mathbf{p}} \\
&\quad + \lambda_{\mathbf{d}}^T[\sigma_{\mathbf{p}} - R_{\mathbf{p}}] + \lambda_{\mathbf{p}}^T[\sigma_{\mathbf{d}} - R_{\mathbf{d}}],
\end{aligned} \tag{5.21}
$$

and we then have, for the primal discrete equations,

$$
\begin{aligned}
\frac{\partial\Phi}{\partial\sigma_{\mathbf{d}}} &= V_{\mathbf{d}}^T S V_{\mathbf{p}}\sigma_{\mathbf{p}} + V_{\mathbf{d}}^T S Z_{\mathbf{p}}\rho_{\mathbf{p}} - V_{\mathbf{d}}^T T_{\mathbf{p}} + \lambda_{\mathbf{p}} = 0, \\
\frac{\partial\Phi}{\partial\rho_{\mathbf{d}}} &= Z_{\mathbf{d}}^T S V_{\mathbf{p}}\sigma_{\mathbf{p}} + Z_{\mathbf{d}}^T S Z_{\mathbf{p}}\rho_{\mathbf{p}} - Z_{\mathbf{d}}^T T_{\mathbf{p}} = 0,
\end{aligned} \tag{5.22}
$$

with a like set for the duals. We can now see that were it not for the third set of relations in (5.18), the σ and ρ would be coupled, but by the use of (5.18), we decouple them and have

$$
\begin{aligned}
&V_{\mathbf{d}}^T S V_{\mathbf{p}}\sigma_{\mathbf{p}} - V_{\mathbf{d}}^T T_{\mathbf{p}} + \lambda_{\mathbf{p}} = 0, \\
&Z_{\mathbf{d}}^T S Z_{\mathbf{p}}\rho_{\mathbf{p}} - Z_{\mathbf{d}}^T T_{\mathbf{p}} = 0,
\end{aligned} \tag{5.23}
$$

with a similar set for the duals. As we shall see later, this decoupling can be carried through when there are multiple cells. The equations for ρ are always cell-by-cell.

It is important to note that, while the distinctness of the several ψ's is maintained for the θ's, the transformation vitiates this distinctness, so that it is no longer possible to associate any component of σ or ρ with any particular ψ. In this sense, the equations in discrete form are solved "all together" and not one-by-one.

6. The multicell case.

6. The multicell case. Once the various quantities in the multivariable case have been relabeled and coalesced into cellwise global assemblies, the treatment of the multicell case follows (formally) almost exactly the treatment of the single-variable multicell case. We therefore reproduce the treatment in [4], indicating when the present case differs from that one.

We partition Ω into K subdomains or cells, $\{\Omega_1, \Omega_2, \ldots, \Omega_K\}$. Between any two *contiguous* cells, say, Ω_k and Ω_m, we have an interface Γ_{km}. In cell discretization, *if two cells are not contiguous, they are not neighbors.* Each cell, say, Ω_k, will now have a boundary made up of more than one segment; we can label these segments by using the notation $\{m[k]\}$ to represent the set of labels for those cells which are contiguous neighbors of Ω_k, say B_k in number.

The vector function $\psi_{\mathbf{p}}(x)$ now becomes an *ensemble* of representations

$$\{\psi_{\mathbf{p}k}(x, \theta_{\mathbf{p}k})\},$$

one for each cell:

$$
\begin{aligned}
\psi_{\mathbf{p}k}(x) &\equiv \{\psi_{\mathbf{p}k\alpha}(x); \alpha = 1, N\} \\
&= \left\{ \sum_{\mu=1}^{M(\alpha)} \phi_{\mathbf{p}k\alpha\mu}(x)\,\theta_{\mathbf{p}k\alpha\mu}; \alpha = 1, N \right\}
\end{aligned} \tag{6.1}
$$

with a similar formula for the dual. The functional Φ_0 is now given as the sum of integrals over the cells

$$\Phi_0 \equiv \sum_{k=1}^{K} \Bigg\{ \int_{\Omega_k} \Big[\nabla\psi_{\mathbf{d}k}^T \cdot \overleftrightarrow{a}_k \cdot \nabla\psi_{\mathbf{p}k} + \psi_{\mathbf{d}k}^T\, \vec{\xi}_k \cdot \nabla\psi_{\mathbf{p}k} + (\nabla\psi_{\mathbf{d}k}^T \cdot \vec{\eta}_k)\,\psi_{\mathbf{p}k} + \psi_{\mathbf{d}k}^T\,(c_k + \nabla \cdot \vec{\eta}_k)\,\psi_{\mathbf{p}k}\Big]\, d\Omega_k - \int_{\Omega_k} \Big[\psi_{\mathbf{p}k}^T\, d_{\mathbf{d}k} + \psi_{\mathbf{d}k}^T\, d_{\mathbf{p}k}\Big]\, d\Omega_k \Bigg\}, \tag{6.2}$$

which combined with (6.1) becomes

$$\Phi_0 = \sum_k \left\{\theta_{\mathbf{d}k}^T\, S_k\, \theta_{\mathbf{p}k} - \theta_{\mathbf{p}k}^T\, T_{\mathbf{d}k} - \theta_{\mathbf{d}k}^T T_{\mathbf{p}k}\right\}, \tag{6.3}$$

with $S_k, T_{\mathbf{p}k}$, and $T_{\mathbf{d}k}$ given by (5.13), except that the formulas are evaluated for each cell.

For each interface Γ_{km}, we may have a separate set of (primal and dual) weight functions $\{w_{km;c\kappa}(s)\}$ and, of course, a separate set of (primal and dual) trace operators. By way of example, the arrays calculated by (5.7) become

$$\begin{aligned} U_{\mathbf{p}km;c\kappa,\alpha\mu} &\equiv \int_\Gamma w_{\mathbf{d}km;c\kappa}(s)\, [\mathcal{U}_{\mathbf{p}km;c\alpha}\phi_{\mathbf{p}k;\alpha\mu}]\,(s)d\Gamma(s), \\ R_{\mathbf{p}km;c\kappa} &\equiv \int_\Gamma w_{\mathbf{d}km;c\kappa}(s) R_{\mathbf{p}km;c}(s)\, d\Gamma(s), \end{aligned} \tag{6.4}$$

with a like set for the duals. Without going into more detail, it is sufficient that the multicell *interface* conditions corresponding to (1.4) are

$$U_{km}\theta_k - R_{km} = U_{mk}\theta_m - R_{mk}, \tag{6.5}$$

with U, θ, and R all appropriate cellwise global arrays. The special case when an "interface" is really a boundary segment is included by assuming that all quantities associated with the exterior of Ω vanish identically (as discussed in [2]).

The complete discretized functional is then

$$\begin{aligned} \Phi_0 = &\sum_k \left\{\theta_{\mathbf{d}k}^T\, S_k\, \theta_{\mathbf{p}k} - \theta_{\mathbf{p}k}^T\, T_{\mathbf{d}k} - \theta_{\mathbf{d}k}^T T_{\mathbf{p}k}\right\} \\ &- \sum_{km} \lambda_{\mathbf{d}km}^T \left\{[U_{\mathbf{p}km}\theta_{\mathbf{p}k} - R_{\mathbf{p}km}] - [U_{\mathbf{p}mk}\theta_{\mathbf{p}k} - R_{\mathbf{p}mk}]\right\} \\ &- \sum_{mk} \lambda_{\mathbf{p}mk}^T \left\{[U_{\mathbf{d}km}\theta_{\mathbf{d}k} - R_{\mathbf{d}km}] - [U_{\mathbf{d}mk}\theta_{\mathbf{d}k} - R_{\mathbf{d}mk}]\right\}. \end{aligned} \tag{6.6}$$

In order to pass from θ to σ and ρ, we must find the appropriate V's and Z's. We note first that cell Ω_k, for example, has not one boundary segment, but B_k. What we have previously referred to as a "global" array is, in fact, defined for each boundary segment $\Gamma_{km[k]}$ of Ω_k. Specifically, the B_k global matrices $\{U_{km[k]}\}$ may be collected into a "superglobal" U_k (with only one subscript) whose rows are the assembled rows of all of the individual coefficient matrices, each associated with a single face of Ω_k. Thus,

$$U_k = \begin{bmatrix} U_{km_1} \\ U_{km_2} \\ \vdots \\ U_{km_{B_k}} \end{bmatrix}. \tag{6.7}$$

We have assumed L^*_{km} moments on Γ_{km}, and M^*_k coefficients $\{\theta_{k\mu^*}\}$, so that U_{km} is an $(L^*_{km} \times M^*_k)$ matrix. U_k, therefore, is in turn an $(L^{**}_k \times M^*_k)$ matrix, where

$$L^{**}_k \equiv \sum_{m[k]} L^*_{km[k]}. \tag{6.8}$$

We assume, as usual, that U_k is nondegenerate, i.e., it has full row rank (requiring that $L^{**}_k \leq M^*_k$), so that we can construct a superglobal V_k and a Z_k using the procedure given in §5. The result is an $(M^*_k \times L^{**}_k)$ matrix V_k and an $(M^*_k \times J^{**}_k)$ matrix Z_k, where $J^{**}_k = M^*_k - L^{**}_k$. We again have

$$\begin{aligned} U_k V_k &= I, \\ U_k Z_k &= 0, \end{aligned} \tag{6.9}$$

with the understanding that the foregoing relations hold for primal and dual quantities. In addition, of course, we adjust the V's for each cell so that

$$V^T_{\mathbf{d}k} S_k Z_{\mathbf{p}k} = 0, \qquad Z^T_{\mathbf{d}k} S_k V_{\mathbf{p}k} = 0. \tag{6.10}$$

Remembering that these are superglobal matrices, we now *repartition* each V_k into a set $\{V_{km[k]}\}$ of submatrices, each with the same number of columns as the corresponding $U_{km[k]}$ has rows. Thus, matching (6.7), we have

$$V_k = [V_{km_1}, V_{km_2}, \ldots, V_{km_{B_k}}]. \tag{6.11}$$

Z_k remains a matrix associated with Ω_k as a whole, and not with any particular face or faces. These new matrices satisfy the following relationships based on (6.9):

$$U_{kp} V_{kq} = \delta_{pq} I, \qquad U_{kp} Z_k = 0, \tag{6.12}$$

for both primal and dual, and

$$V^T_{\mathbf{d}km} S_k Z_{\mathbf{p}k} = 0, \qquad Z^T_{\mathbf{d}k} S_k V_{\mathbf{p}km} = 0. \tag{6.13}$$

We are now ready to change variables. With the exception that the matrices are superglobal, as opposed to merely global, the treatment is identical to that in [4]. The transformation corresponding to (5.19) is

$$\theta_k = \sum_{p[k]} V_{kp}(\sigma_{kp} + R_{kp}) + Z_k \rho_k, \tag{6.14}$$

where the summation is, as the notation expresses, over the labels of the contiguous neighbors of Ω_k. When we apply the discrete trace operator U_{km} to θ_k, we obtain

$$\begin{aligned} U_{km}\theta_k &= \sum_{p[k]} U_{km} V_{kp}(\sigma_{kp} + R_{kp}) + U_{km} Z_k \rho_k \\ &= \sum_{p[k]} \delta_{mp}(\sigma_{kp} + R_{kp}) + 0 \cdot \rho_k \\ &= \sigma_{km} + R_{km} \end{aligned} \tag{6.15}$$

with a matching reduction for $U_{mk}\theta_m$. Equation (6.5) then reduces to

$$(\sigma_{km} + R_{km}) - R_{km} = (\sigma_{mk} + R_{mk}) - R_{mk} \tag{6.16}$$

or

$$\sigma_{km} = \sigma_{mk} \tag{6.17}$$

for primal and dual quantities.

Next, we substitute (6.14) into (6.5) (with a change of index names) to obtain the rather lengthy result:

$$\begin{aligned}\Phi = &\sum_{q=1}^{K} [\Phi_{\sigma\sigma,q} + \Phi_{\rho\rho,q} + \Phi_{\sigma\rho,q} + \text{constant}] \\ &-\sum_{\substack{q=1\\r=1}}^{K} \lambda_{\mathbf{d}qr}(\sigma_{\mathbf{p}qr} - \sigma_{\mathbf{p}rq}) - \sum_{\substack{q=1\\r=1}}^{K} \lambda_{\mathbf{p}qr}(\sigma_{\mathbf{d}qr} - \sigma_{\mathbf{d}rq})\end{aligned} \tag{6.18}$$

with

$$\begin{aligned}\Phi_{\sigma\sigma,q} = &\sum_{\substack{r[q]\\s[q]}} \left[\sigma^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q V_{\mathbf{p}qs} \sigma_{\mathbf{p}qs} + \sigma^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q V_{\mathbf{p}qs} R_{\mathbf{p}qs} \right. \\ &\left. \qquad + R^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q V_{\mathbf{p}qs} \sigma_{\mathbf{p}qs} \right] \\ &- \sum_{r[q]} \left[\sigma^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} T_{\mathbf{p}q} + T^T_{\mathbf{d}q} V_{\mathbf{p}qr} \sigma_{\mathbf{p}qr} \right],\end{aligned} \tag{6.19}$$

$$\begin{aligned}\Phi_{\rho\rho,q} = &\rho^T_{\mathbf{d}q} Z^T_{\mathbf{d}q} S_q Z_{\mathbf{p}q} \rho_{\mathbf{p}q} + \sum_{r[q]} \left[R^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q Z_{\mathbf{p}q} \rho_{\mathbf{p}q} + \rho^T_{\mathbf{d}q} Z^T_{\mathbf{d}q} S_q V_{\mathbf{p}qr} R_{\mathbf{p}qr} \right] \\ &- \rho^T_{\mathbf{d}q} Z^T_{\mathbf{d}q} T_{\mathbf{p}q} - T^T_{\mathbf{d}q} Z_{\mathbf{p}q} \rho_{\mathbf{p}q},\end{aligned} \tag{6.20}$$

and

$$\Phi_{\sigma\rho,q} = \sum_{r[q]} \left[\sigma^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q Z_{\mathbf{p}q} \rho_{\mathbf{p}q} + \rho^T_{\mathbf{d}q} Z^T_{\mathbf{d}q} S_q V_{\mathbf{p}qr} \sigma_{\mathbf{p}qr} \right], \tag{6.21}$$

and the understanding that the double summations are only over values of q and r that label contiguous neighbors.

We can ignore the constant part because we are going to differentiate Φ with respect to the variables σ and ρ. It is very advantageous to decouple σ and ρ, so that the discrete equation for one will not involve the other. We shall see that this will confer the very attractive feature of being able to do almost everything cell-by-cell, as mentioned already in §5. The term interfering with this is $\Phi_{\sigma\rho,q}$, so we must make this vanish. All that is necessary is to choose the V's so as to satisfy

$$V^T_{\mathbf{d}qr} S_q Z_{\mathbf{p}q} = 0, \qquad Z^T_{\mathbf{d}q} S_q V_{\mathbf{p}qr} = 0. \tag{6.22}$$

Bearing in mind that r *labels only the contiguous neighbors* of Ω_q, i.e., the *faces of* Ω_q, it is clear that the conditions (6.22) refer to *each individual cell* Ω_q in turn. Hence, we can treat each cell exactly as in §5, i.e., group the $\{U_{qr}\}$ into one matrix, find the appropriate V and Z, and repartition the V, etc. Note also that the matrices chosen in this way cause the bracket in (6.20) to vanish.

To condense the formulas, we define

$$H_{qrs} \equiv V^T_{\mathbf{d}qr} S_q V_{\mathbf{p}qs}, \qquad A_q \equiv Z^T_{\mathbf{d}q} S_q Z_{\mathbf{p}q}, \tag{6.23}$$

(6.24)
$$G_{\mathbf{p}qr} \equiv V_{\mathbf{d}qr}^T T_{\mathbf{p}q} - \sum_{s[q]} H_{qrs} R_{\mathbf{p}qs},$$
$$G_{\mathbf{d}qr} \equiv V_{\mathbf{p}qr}^T T_{\mathbf{d}q} - \sum_{s[q]} H_{qrs}^T R_{\mathbf{d}qs},$$

and

(6.25)
$$J_{\mathbf{p}q} \equiv Z_{\mathbf{d}q}^T T_{\mathbf{p}q}, \qquad J_{\mathbf{d}q} \equiv Z_{\mathbf{p}q}^T T_{\mathbf{d}q}.$$

Then we have, for Φ,

(6.26)
$$\begin{aligned}\Phi = \sum_{q=1}^{K} \Bigg\{ \sum_{\substack{r[q]\\ s[q]}} \sigma_{\mathbf{d}qr}^T H_{qrs} \sigma_{\mathbf{p}qs} - \sum_{r[q]} \left[\sigma_{\mathbf{d}qr}^T G_{\mathbf{p}qr} + G_{\mathbf{d}qr}^T \sigma_{\mathbf{p}qr}\right] \\ + \rho_{\mathbf{d}q}^T A_q \rho_{\mathbf{p}q} - \rho_{\mathbf{d}q}^T J_{\mathbf{p}q} - J_{\mathbf{d}q}^T \rho_{\mathbf{p}q} \Bigg\} \\ + \sum_{\substack{q=1\\ r=1}}^{K} \lambda_{\mathbf{d}qr}(\sigma_{\mathbf{p}qr} - \sigma_{\mathbf{p}rq}) + \sum_{\substack{q=1\\ r=1}}^{K} \lambda_{\mathbf{p}qr}(\sigma_{\mathbf{d}qr} - \sigma_{\mathbf{d}rq}).\end{aligned}$$

To obtain the discrete equations for σ and ρ, we first differentiate with respect to $\rho_{\mathbf{d}k}$. (With the changed indices, it is much easier to follow the procedure correctly.) We obtain, for the primal,

(6.27)
$$A_k \rho_{\mathbf{p}k} = J_{\mathbf{p}k}.$$

Obviously, each ρ_k can be found for Ω_k independently of any other cell. This part of the solution can therefore readily be parallelized.

We next differentiate Φ with respect to $\sigma_{\mathbf{d}km}$. The result is

(6.28)
$$\sum_{s[k]} H_{kms} \sigma_{\mathbf{p}ks} - G_{\mathbf{p}km} + (\lambda_{\mathbf{p}km} - \lambda_{\mathbf{p}mk}) = 0.$$

The antisymmetry of the interface conditions is reflected in the antisymmetry of the combined λ terms. We make use of this important fact to eliminate the λ's. We interchange k and m in (6.28) and add the result back to (6.28). The λ terms cancel and we are left with

(6.29)
$$\sum_{s[k]} H_{kms} \sigma_{\mathbf{p}ks} + \sum_{s[m]} H_{mks} \sigma_{\mathbf{p}ms} - G_{\mathbf{p}km} - G_{\mathbf{p}mk} = 0.$$

The block diagonal part may be extracted from (6.29) by withdrawing the term with $s[k] = m$ from the first sum and the term with $s[m] = k$ from the second. Since $\sigma_{\mathbf{p}mk} = \sigma_{\mathbf{p}km}$, we get

(6.30)
$$\Lambda_{km} \sigma_{\mathbf{p}km} + {\sum_{s[k]}}' H_{kms} \sigma_{\mathbf{p}ks} + {\sum_{s[m]}}' H_{mks} \sigma_{\mathbf{p}ms} - G_{\mathbf{p}km} - G_{\mathbf{p}mk} = 0$$

with

(6.31)
$$\Lambda_{km} \equiv H_{kmm} + H_{mkk},$$

and the primes on the summations mean that the km and mk terms are left out. Equations (6.27) and (6.30) are the discrete equations to be solved. This was done by Gaussian elimination, since all of these equation systems are small for the problems to be discussed.

7. Numerical tests. All the problems are set within the square $(-1 \leq x \leq 1; -1 \leq y \leq 1)$. Unless otherwise indicated, it is divided into four equal square cells. All the problems were run with three or four moments per boundary or interface condition, because experience has shown that this generally gives good results for two-dimensional problems. The number of degrees of freedom within each cell was chosen so as to remove interface condition degeneracies. The basis functions and the weights were Legendre polynomials.

PROBLEM 1. A gradient-divergence problem. One of the simplest systems equivalent to Laplace's equation is

$$\vec{v} + \nabla p = 0, \qquad \nabla \cdot \vec{v} = 0 \tag{7.1}$$

in two dimensions. By taking the divergence of the first equation and subtracting the second, it is clear that p satisfies Laplace's equation. We can compare the results of solving a suitable boundary value problem in two different ways: as a single, second-order PDE, or as the system (7.1). It is not always clear what the interface conditions should be for the latter problem, even when those for the former are very well established. In this case, the single-variable problem serves as a good guide to the system problem.

To begin, we assume p to be equal to the real part of the analytic function $z^3+1.5z$, i.e., $p = x^3 - 3xy^2 + 1.5x$, and set the Dirichlet boundary conditions accordingly. The interface conditions are that p be continuous (in the weak sense of having the first four moments of the discrepancy vanish) across the interfaces. Similarly, the first four moments of the solution on the boundary must match those of the boundary data.

There are 36 degrees of freedom in each cell (seventh degree), of which $4 \times 4 = 16$ go to satisfy the boundary and interface conditions. The remaining 20 are free to make the representation of the unknown solution satisfy the PDE within each cell as closely as possible. The discrete equations (6.27) for the latter variables are therefore 20 in number for each cell and are independent of each other, as shown in §6. The degrees of freedom on each interface are four in number, so that there are $4 \times 4 = 16$ equations (6.30), also independent of the intracell equations. Thus, as indicated at the beginning of this section, the equations systems are all small and are easily solved directly. The solution comes out exactly.

The simplest functional whose primal Euler equation is (7.1) is

$$\Phi_0 \equiv \int_\Omega \{\vec{v}_{\mathrm{d}} \cdot (\vec{v}_{\mathrm{p}} + \nabla p_{\mathrm{p}}) + p_{\mathrm{d}}(\nabla \cdot \vec{v}_{\mathrm{p}})\}\, d\Omega, \tag{7.2}$$

and the simplest interface condition is to require the continuity of p, as for the single-variable case. Also, the value of p is specified on the boundary as before. With these specifications in the computer program, the solution found is shown in Fig. 1. The result is very poor, and can obviously be related to the lack of smoothness across the interfaces.

The difficulty is that, while in the single-variable case the variational process gives rise to *induced* (natural) interface conditions (as discussed in [3]), the system (7.1) is first-order, so that it is possible to have no induced interface conditions, as happens for (7.2). The reason for this is suggested by (2.4), in which an induced surface condition is shown, *but only if* $\vec{\eta} \neq 0$, which is not the case for (7.2), since no derivative of a dual variable appears. To remedy this, an alternative form for the functional may be chosen, e.g.,

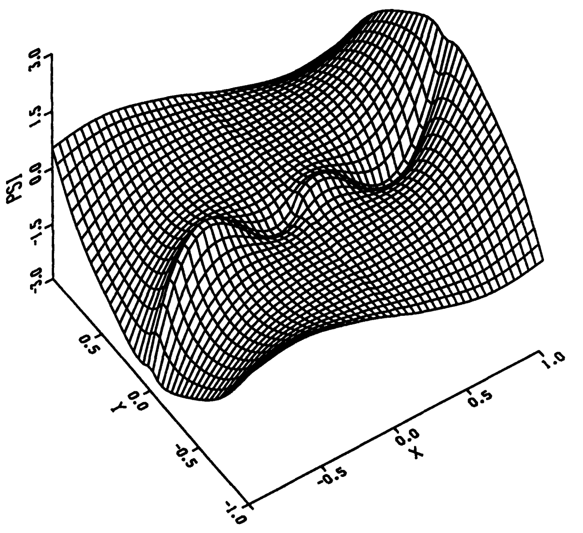


FIG. 1. *Gradient-divergence problem. Solution with simplest functional.*

$$\Phi_0 \equiv \int_\Omega \{\vec{v}_\mathrm{d} \cdot (\vec{v}_\mathrm{p} + \nabla p_\mathrm{p}) - \nabla p_\mathrm{d} \cdot \vec{v}_\mathrm{p}\}\, d\Omega, \tag{7.3}$$

which causes a surface term containing $\delta p_\mathrm{d} v_{\mathrm{p}n}$ to be induced on each side of each interface. The difference between these terms for cells Ω_k and Ω_m, for example, can be shown to vanish (in a particular sense; see [4, §6]). Roughly, it means that on Γ_{km}, $\delta p_{\mathrm{d}k}(v_{\mathrm{p}k})_n = \delta p_{\mathrm{d}m}(v_{\mathrm{p}m})_n$. If, therefore, $\delta p_{\mathrm{d}k} = \delta p_{\mathrm{d}m}$, then $(v_{\mathrm{p}k})_n = (v_{\mathrm{p}m})_n$. Thus the normal component of v_p is made weakly continuous across Γ_{km} and this, in turn, is equivalent to making $\partial p_\mathrm{p}/\partial n$ weakly continuous across Γ_{km}, as for the single-variable case.

The result of the computation bears out this reasoning. p again comes out exactly, and looks as shown in Fig. 2. The sizes of the equation systems are as follows: the number of degrees of freedom per cell is $3 \times 28 = 84$, of which $4 \times 4 = 16$ are used up on perimeter constraints, leaving 68, which is the size of the equation system in each cell. Since there are four constraints per interface, there is a total of $4 \times 4 = 16$ equations (6.30).

A set of boundary data (a zig-zag function) for which an exact solution is unknown leads to a solution of (7.1) shown in Fig. 3. The reference solution of the Laplace equation differs by about 1 percent.

PROBLEM 2. The Cauchy–Riemann equations. This system of first-order equations in the variables (u, v), which may represent the real and imaginary parts of a complex function, characterizes the analyticity of that function. It is one of the simplest (but also one of the most difficult) elliptic systems in two variables and is, in fact, the principal part of a canonical form for all linear elliptic first-order systems in two variables. The equations are

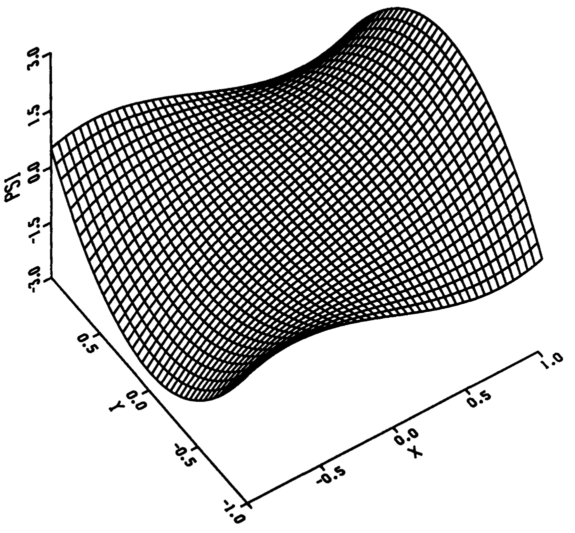


FIG. 2. *Gradient-divergence problem. Solution with altered functional.*

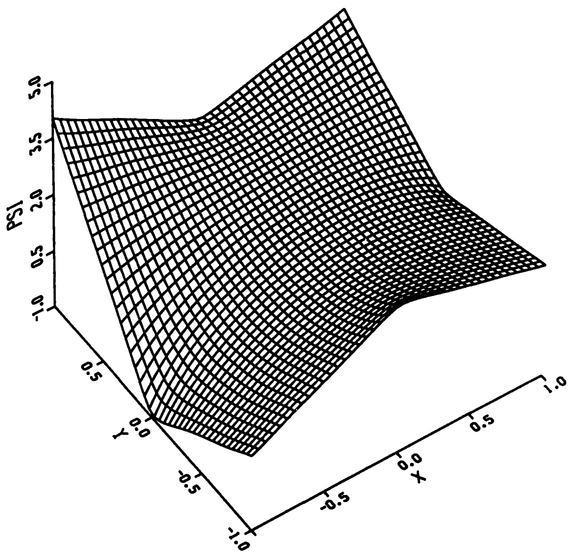


FIG. 3. *Gradient-divergence problem. Solution with zig-zag boundary conditions.*

$$\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} = 0, \qquad \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = 0. \tag{7.4}$$

As is done in fluid dynamical studies, it is also useful to consider the couple $(u, -v)$ as a vector $\vec{w}$ which, because of the second equation of (7.4), is the gradient of a potential function $\psi(x, y)$. Thus

$$u = \frac{\partial \psi}{\partial x}, \qquad -v = \frac{\partial \psi}{\partial y}. \tag{7.5}$$

The first equation, in turn, results in ψ satisfying Laplace's equation. For this equation, appropriate boundary conditions are that ψ or $\partial\psi/\partial n$ (or a linear combination of the two) are to be equal to some boundary data. Since we wish to treat (7.4) strictly as a system in terms of u and v, we must choose a Neumann condition involving $\partial\psi/\partial n$ for the boundary condition. We note that since $\vec{w} = \nabla\psi$, we have

$$\frac{\partial \psi}{\partial n} = \vec{n}\cdot\nabla\psi = \vec{n}\cdot\vec{w} = w_n, \tag{7.6}$$

so that u is to be specified on the vertical sides of Ω, and $-v$ on the horizontal sides. As regards the interfaces, the same considerations apply; we must require that the normal component of $\vec{w}$ be (weakly) continuous across each interface.

The first, very simple problem tried has the solution $u = x, v = y$, i.e., the real and imaginary parts of the complex variable z. The boundary conditions are

$$\begin{aligned} u(-1,y) &= -1, \qquad & u(1,y) &= 1, \\ v(x,-1) &= -1, \qquad & v(x,1) &= 1. \end{aligned} \tag{7.7}$$

The solution is exactly the correct one: $(u,v) = (x,y)$.

The next problem is to find the solution $(u,v) = (x^2 - y^2, 2xy)$, which represents the real and imaginary parts of z^2. The boundary conditions are

$$\begin{aligned} u(-1,y) &= 1 - y^2, \qquad & u(1,y) &= 1 - y^2, \\ v(x,-1) &= -2x, \qquad & v(x,1) &= 2x. \end{aligned} \tag{7.8}$$

The solutions for u and v are unfortunately incorrect; they are shown in Figs. 4(a) and 4(b).

Various alternative ways of posing the problem were tried in an effort to make it give the right answer. Both u and v were forced to be weakly continuous across the interfaces, small additional terms were added to the equations, the terms in the functional were changed so as to induce interface conditions, and various alternative forms of the boundary conditions were applied. The only alteration that yielded the correct answer was the addition of a small multiple of the Laplacian of each variable to the corresponding equation. The actual system solved was

$$-\epsilon\nabla^2 u + \frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} = 0, \qquad -\epsilon\nabla^2 v + \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = 0, \tag{7.9}$$

with $\epsilon = 10^{-}5$, and this yielded the correct solution to 10 figures. It is not clear why this should be so, but it has been noticed that the presence of at least one singular diagonal block Λ_{km} is enough to give rise to an incorrect solution. Adding the second-order terms prevents this from happening. The correct solution is shown in Figs. 5(a) and 5(b).

The orders of the intracell and interface linear systems were 30 and 12, respectively. (Of course, there were four intracell sets.)

PROBLEM 3. The Stokes problem. The linearized Navier–Stokes equations for the flow of an incompressible fluid are (see, e.g., [6])

$$-\nabla^2\vec{v} + \nabla p = \vec{f}, \qquad \nabla\cdot\vec{v} = 0, \tag{7.10}$$

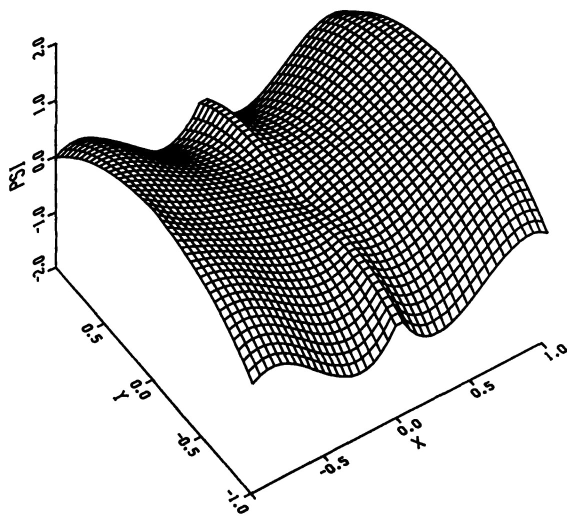


FIG. 4(a). *Cauchy–Riemann equations. Solution for u with no additional terms in functional.*

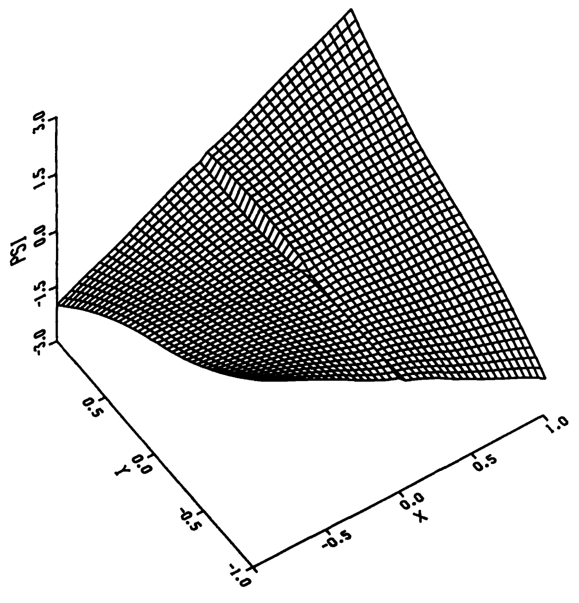


FIG. 4(b). *Cauchy–Riemann equations. Solution for v with no additional terms in functional.*

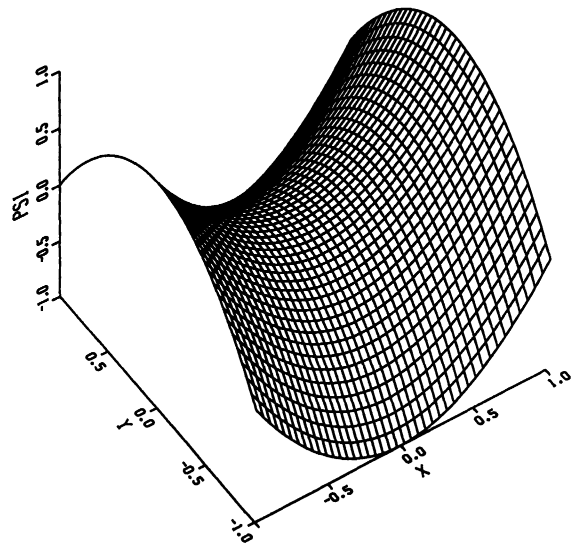


FIG. 5(a). *Cauchy–Riemann equations. Solution for u with additional terms.*

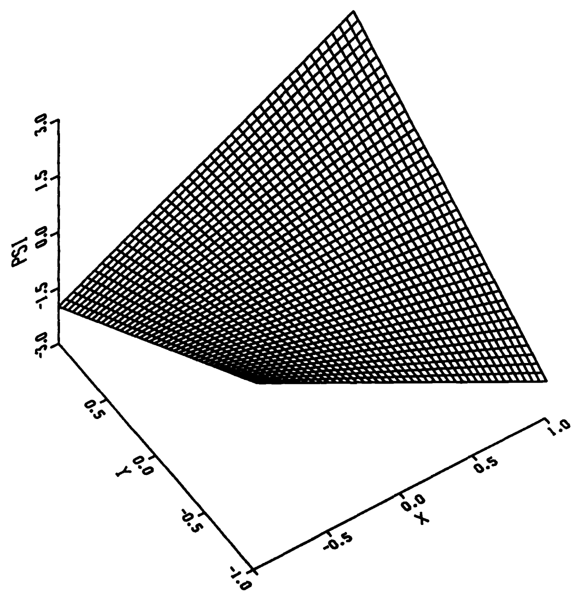


FIG. 5(b). *Cauchy–Riemann equations. Solution for v with additional terms.*

with the velocity $\vec{v}$ specified on the boundary. The pressure p is determined to within an additive constant.

The simplest functional was tried initially, viz.,

$$\Phi_0 = \int_\Omega \{\nabla\vec{v}_\mathbf{d} : \nabla\vec{v}_\mathbf{p} + \vec{v}_\mathbf{d} \cdot \nabla p_\mathbf{p} + p_\mathbf{d}\nabla \cdot \vec{v}_\mathbf{p}\}\, d\Omega. \tag{7.11}$$

The double-dot symbol between the two (diadic) gradients of the primal and dual velocities means that the two gradient operators are first dotted together, followed by the two velocity vectors (this is the same as the double contraction of the four-index tensor formed from the derivatives of $\vec{v}_\mathbf{d}$ and $\vec{v}_\mathbf{p}$).

The first case attempted calls for $\vec{v}_\mathbf{p} = (1, 1)$ on the boundary. There is no condition on $p_\mathbf{p}$. (Corresponding boundary conditions are applied to the dual variables.) As usual, the question arises as to what the proper interface conditions should be. The most obvious choice is again to apply the same kind of conditions at the interface as are applied at the boundary of a single domain. Therefore, the two components of $\vec{v}_\mathbf{p}$ are made weakly continuous across the interfaces. (The same condition is applied to $\vec{v}_\mathbf{d}$.) This choice yields the exact solution for the velocity components, viz., $\vec{v}_\mathbf{p} = (1, 1)$ throughout Ω.

On the other hand, the pressure, which should have the value $p(x, y) = x + y +$ constant, has nothing to force it to be continuous across the interfaces, so it comes out as shown in Fig. 6. By adding the appropriate constant to each of three cells, so as to relate them properly to the first, the global solution can be made continuous. This is not, however, a satisfactory solution. We might force weak continuity of p across the interfaces, but this results in three boundary conditions for each internal cell face, and this is too many. A resolution of this dilemma is to *induce* the constraint on p by changing the second term of the functional (7.11) into its adjoint, as was done with (7.3). The result is

$$\Phi_0 = \int_\Omega \{\nabla\vec{v}_\mathbf{d} : \nabla\vec{v}_\mathbf{p} - (\nabla \cdot \vec{v}_\mathbf{d})p_\mathbf{p} + p_\mathbf{d}\nabla \cdot \vec{v}_\mathbf{p}\}\, d\Omega, \tag{7.12}$$

so that the induced surface term resulting from the variation of the dual quantities is

$$\Phi_S = \int_\Gamma -\delta v_{\mathbf{d}n} p_\mathbf{p} d\Gamma. \tag{7.13}$$

The usual variational interface argument shows that if $\delta v_{\mathbf{d}n}$ is continuous across the interface, then so is $p_\mathbf{p}$. Hence, since continuity of $v_{\mathbf{d}n}$ is imposed (the dual variables have the same interface and boundary conditions as the primal), $p_\mathbf{p}$ should be weakly continuous. However, as Fig. 7 shows, this is not enough to give the correct solution. It was also necessary to add a small multiple of p to the second equation of (7.10). This combination of the altered functional and the small added term yields the correct continuous solution $p_\mathbf{p}$, as shown in Fig. 8, besides giving the correct result for the velocity, all to six figures, when the multiple of p was 10^{-7}.

Using this same setup of functionals and constraints, a somewhat more complicated example was tried. The boundary data were represented by cubic and quadratic polynomials, and care was taken to maintain the incompressibility at the corners, where the v-derivatives involved can actually be calculated from the given functions. Four moments were used in the collocation, in order to match the cubic boundary

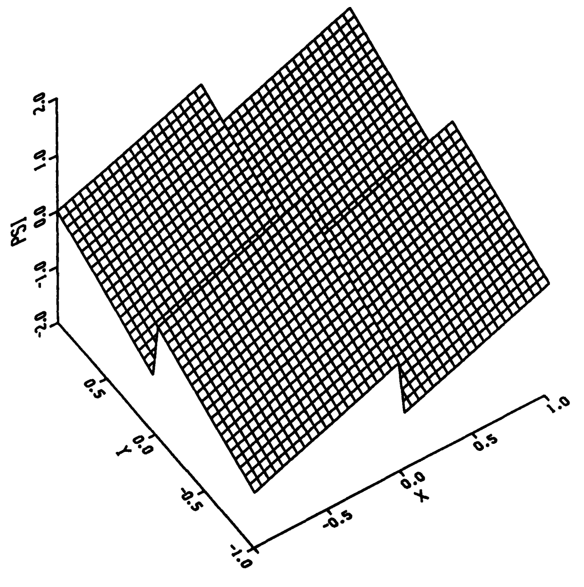


FIG. 6. *Stokes problem, flat velocity. Pressure solution with simplest functional.*

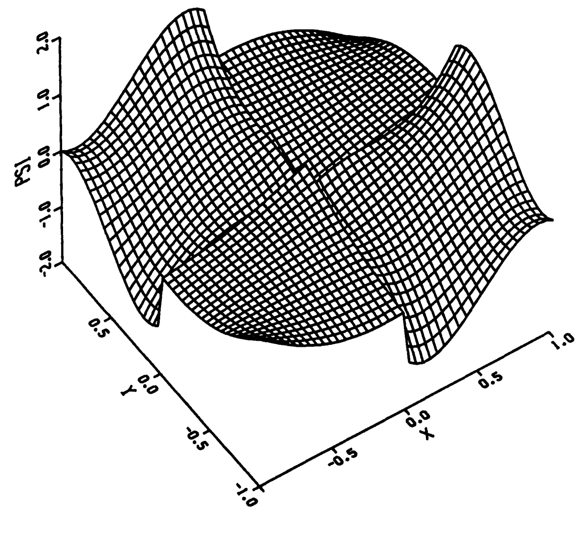


FIG. 7. *Stokes problem, flat velocity. Pressure solution with corrected functional.*

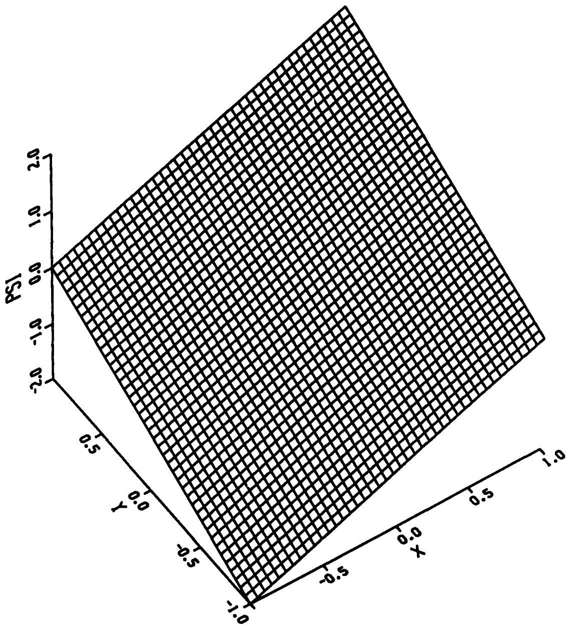


FIG. 8. *Stokes problem, flat velocity. Pressure solution with corrected functional and additional terms.*

data functions. The results for the velocity components are quite smooth, as can be seen in Figs. 9(a) and 9(b). The pressure is very sensitive to even slight irregularities in the velocity components, because it is related to their derivatives, but the result for the pressure is still satisfactory, as can be seen in Fig. 9(c).

The orders of the intracell and interface linear systems were 39 and 24, respectively, except for the wavy case, which required more moment collocations. In that case, the orders were 84 and 32.

PROBLEM 4. A clamped plate. Our last example consists of the solution of the equation

$$\nabla^4 u = d \tag{7.14}$$

with $d = 1$, and subject to the boundary conditions

$$u(s) = \frac{\partial u}{\partial n}(s) = 0, \tag{7.15}$$

with the second condition representing the clamping. In order to make the equations for the problem conform to (1.3), we introduce the auxiliary variable v and replace (7.14) by

$$-\nabla^2 u + v = 0, \qquad -\nabla^2 v = -d, \tag{7.16}$$

for which the simplest functional (in primal-dual form) is

$$\Phi_0 = \int_\Gamma \{\nabla u_\mathrm{d} \cdot \nabla u_\mathrm{p} + \nabla v_\mathrm{d} \cdot \nabla v_\mathrm{p} + u_\mathrm{d} v_\mathrm{p} + v_\mathrm{d} d\}\, d\Omega. \tag{7.17}$$

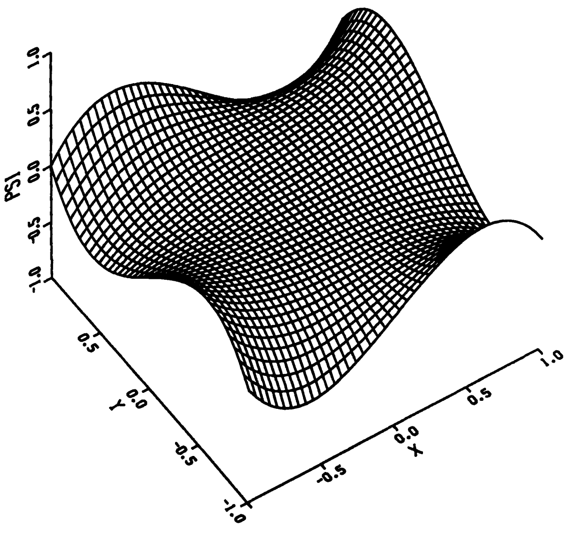


FIG. 9(a). *Stokes problem, wavy velocity. u-solution with corrected functional and additional terms.*

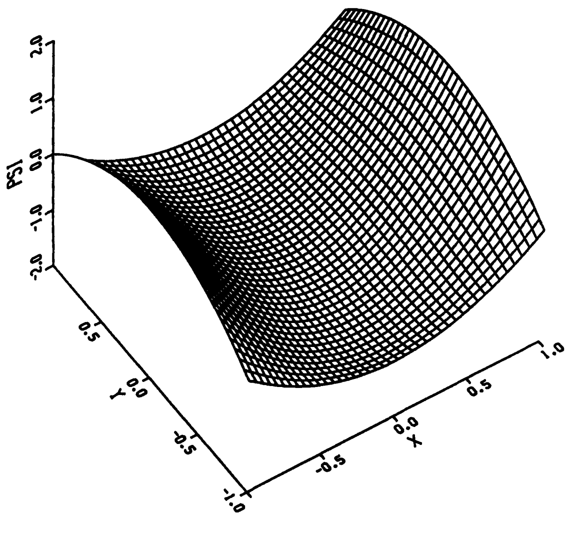


FIG. 9(b). *Stokes problem, wavy velocity. v-solution with corrected functional and additional terms.*

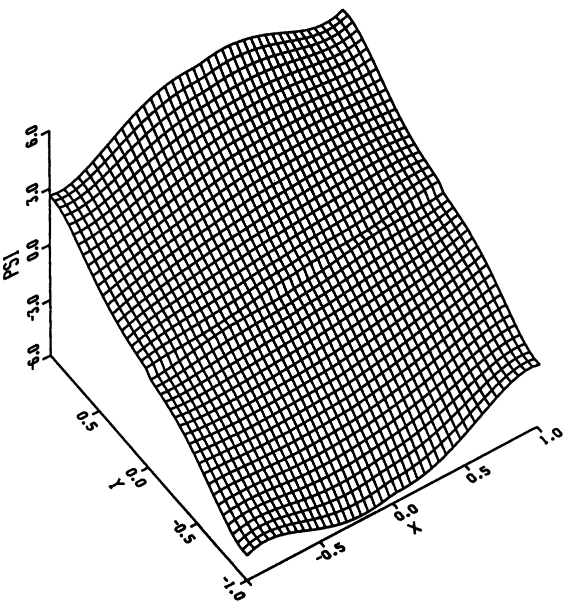


FIG. 9(c). *Stokes problem, wavy velocity. Pressure solution with corrected functional and additional terms.*

With these choices, the solution is not correct. Figure 10 shows the result for u_p with four cells and three moments on each interface, and with a sixth degree polynomial in each cell. On the other hand, when the equations (7.16) are *reordered*, we obtain the accurate result for u_p shown in Fig. 11. (In the last two figures, the ordinates have been rescaled to improve the appearance of the plots.)

The argument for exchanging the equation order is basically that the functional (7.17) is not unique, depending as it does on the association of the first equation in (7.16) with u_d and the second with v_d. Reversing the order of the equations amounts to interchanging u_d and v_d in the functional.

A rough idea of the error can be inferred by noting that the maximum value of u over the plotting grid is .020237 for the 4-cell case and .020245 for a 16-cell case (with otherwise all parameters the same), yielding a difference of .000008.

The orders of the intracell and interface linear systems for the four-cell case were 48 and 24, respectively.

8. Conclusions.

The cell discretization algorithm has proved successful in solving several quite different model problems involving systems of equations in more unknowns than one. The treatment of the discrete equations involves grouping the degrees of freedom (the θ's) for all of the unknowns into one global array for each cell, and transforming this vector to the usual two sets of auxiliary unknowns (the σ's and the ρ's) without associating them with particular unknowns. By this means, the usual auxiliary matrices (V's and Z's) can be constructed in a straightforward manner so that the σ's and ρ's are completely decoupled, and can be solved for separately. It

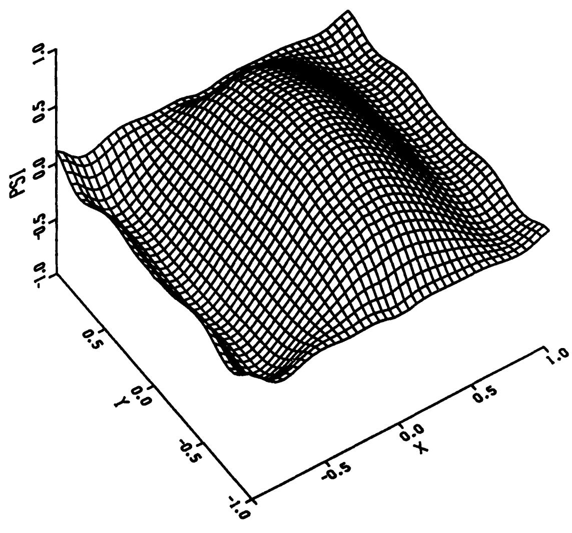


FIG. 10. *Clamped-plate, u-solution. Wrong equation ordering.*

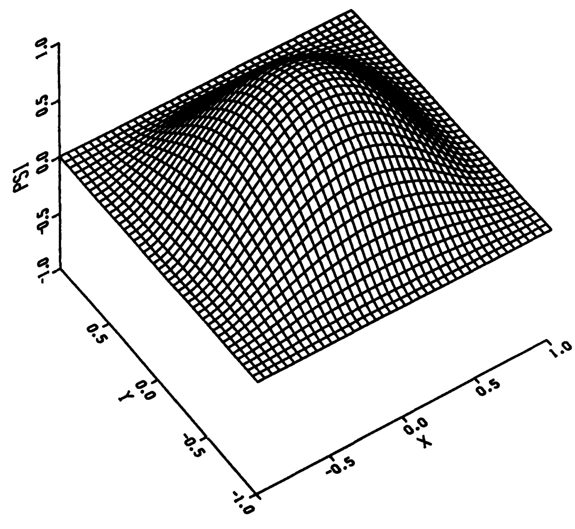


FIG. 11. *Clamped-plate, u-solution. Correct equation ordering.*

follows that the calculation can be parallelized almost completely with no artificial reshuffling of the variables.

Gauss elimination was used to solve the linear systems of equations for the intracell and interface variables with satisfactory results because the equation systems for these problems are all of modest size. A few unexpected complications arose, mostly due to the singular nature of the original problems. It was found necessary to add small additional terms to the original equations in order to render the discrete equations less degenerate. The question of what is the correct functional corresponding to a given (nonselfadjoint) set of PDEs therefore remains open. Moreover, a careful selection of the interface conditions was found to be necessary, coupled with the appropriate choice of terms in the functional used. However, there was nothing unnatural or bizarre in these choices.

Although the problems solved were very simple ones, they still contained some challenging complexities. It is encouraging that the CD method was capable of solving them without any basic modification. The major shortcoming of the computer program used for these computations is the absence of the capability of using "special" representations (e.g., nonpolynomial, general algebraic, and transcendental functions) for the solution approximations, which are nonlinear in the degrees of freedom. (This shortcoming has since been remedied.)

Acknowledgment. I am deeply indebted to Professor M.J.D. Powell for his kindness and forbearance in giving me the continuing opportunity to pursue this research in his Numerical Analysis Group in the Department of Applied Mathematics and Theoretical Physics at Cambridge University (as well as for his clarifying a matrix problem for me). Thanks are also due to Professor N. Sugimoto for several very helpful discussions, as well as to Drs. H. Yee and H. Lomax for providing me with the facilities at the NASA–Ames Research Center needed to revise this paper.

REFERENCES

[1] M. Gelfand and S.V. Fomin, *Calculus of Variations*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[2] J. Greenstadt, *The cell discretization algorithm for elliptic partial differential equations*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 261–288.

[3] ———, *On the reconciliation of clashing boundary conditions in cell discretization*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1282–1306.

[4] ———, *Cell discretization of nonselfadjoint linear elliptic partial differential equations*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1074–1108.

[5] ———, *Minimum rank boundary corrections for nonselfadjoint p.d.e.'s in cell discretization*, DAMPT Report/1989/NA1, Numerical Analysis Group, Dept. of Appl. Maths. and Theoret. Phys., Univ. of Cambridge, U.K., 1989.

[6] O. A. Ladyzhenskaya, *The Mathematical Theory of Viscous Incompressible Flow*, Gordon and Breach, New York, 1963.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 654–669, May 1993 010

A PARALLEL GRAPH COLORING HEURISTIC*

MARK T. JONES† AND PAUL E. PLASSMANN†

Abstract. The problem of computing good graph colorings arises in many diverse applications, such as in the estimation of sparse Jacobians and in the development of efficient, parallel iterative methods for solving sparse linear systems. This paper presents an asynchronous graph coloring heuristic well suited to distributed memory parallel computers. Experimental results obtained on an Intel iPSC/860 are presented, which demonstrate that, for graphs arising from finite element applications, the heuristic exhibits scalable performance and generates colorings usually within three or four colors of the best-known linear time sequential heuristics. For bounded degree graphs, it is shown that the expected running time of the heuristic under the P-RAM computation model is bounded by $EO(\log(n)/\log\log(n))$. This bound is an improvement over the previously known best upper bound for the expected running time of a random heuristic for the graph coloring problem.

Key words. distributed memory computers, graph coloring heuristics, parallel algorithms, random algorithms, sparse matrices

AMS(MOS) subject classifications. 65F10, 65F50, 65Y05, 68Q22, 68R10

1. Introduction.

The determination of the chromatic number of a general graph is a well-known NP-hard problem [4]. However, a number of practical problems require the determination of nearly optimal graph colorings. For example, it has been shown [3] that the problem of directly estimating a sparse Jacobian by finite differences with a minimum number of function evaluations is equivalent to a graph coloring problem. Also, it has been shown [10] that the minimum number of parallel steps in the solution of the triangular systems involving incomplete Cholesky factors can be obtained by a matrix reordering derived from the solution of a graph coloring problem. Thus, the development of an effective parallel heuristic is of great practical, as well as theoretical, interest.

This paper presents an asynchronous graph coloring heuristic well suited to distributed memory parallel computers. Our heuristic consists of two phases: an initial, parallel phase that uses a random heuristic, followed by a local phase that utilizes one of several good sequential coloring heuristics. The initial phase maintains the good expected running time bounds obtained for a Monte Carlo algorithm for determining a maximal independent set [8]. In fact, for bounded degree graphs, we show that an upper bound for the expected running time of our heuristic under the P-RAM computation model is $EO(\log(n)/\log\log(n))$. This bound is an improvement over the previously known best upper bound of $EO(\log(n))$ for a random heuristic.

To illustrate the performance of this heuristic, we present experimental results obtained on an Intel iPSC/860. The results demonstrate that, for graphs arising from finite element applications, the heuristic exhibits scalable performance. In addition, for these problems the heuristic is shown to generate colorings usually using no more than three or four more colors than the best linear time sequential heuristics.

This paper is organized as follows. In §2 we introduce a new Monte Carlo heuristic that eliminates the need for global synchronization of the processors and allows for the more efficient use of data structures. We prove, in §3, that under the P-RAM computational model the expected running time of the asynchronous heuristic

*Received by the editors September 30, 1991; accepted for publication (in revised form) May 11, 1992. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

†Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

affords an improvement over the upper bound for a Monte Carlo algorithm based on finding maximal independent sets. A distributed memory implementation of the asynchronous heuristic is detailed in §4. In §5 we present and discuss experimental results obtained on the iPSC/860. Finally, in §6 we discuss possible avenues for improving the heuristic.

2. An asynchronous parallel graph coloring heuristic.

In this section we introduce a graph coloring heuristic suitable for asynchronous parallel machines. First, we review the graph coloring problem. Consider the symmetric graph $G = (V, E)$ with vertex set V, with $|V| = n$, and edge set E. We say that the function $\sigma : V \rightarrow \{1, \ldots, s\}$ is an s-coloring of G, if $\sigma(v) \neq \sigma(w)$ for all edges $(v, w) \in E$. The minimum possible value for s is known as the *chromatic number* of G, which we denote as $\chi(G)$.

The question as to whether a general graph G is s-colorable is NP-complete [4]. It is known that unless $P = NP$, there does not exist a polynomial approximation scheme for solving the graph coloring problem [4]. In fact, the best polynomial time heuristic known [6] can theoretically guarantee a coloring of only size $c\,(n/\log n)\,\chi(G)$, where c is some constant.

Given these pessimistic theoretical results, it is quite surprising that, for certain classes of graphs, there exist a number of sequential graph coloring heuristics that are very effective in practice. For graphs arising from a number of applications, it has been demonstrated that these heuristics are often able to find colorings that are within one or two of an optimal coloring [3], [7].

```
$V' \leftarrow V$;
**For** $i = 1, \ldots, n$ **do**
    Choose a vertex $v_i$ from $V'$;
    $\sigma(v_i)$ = smallest available consistent color;
    $V' \leftarrow V' \setminus \{v_i\}$;
**enddo**
```

FIG. 1. *A sequential greedy coloring heuristic.*

It is known that an optimal coloring can be obtained with a greedy heuristic if the vertices are visited in the correct order [1]. The basic structure of the greedy heuristic is shown in Fig. 1. The only aspect of the sequential heuristic that must be specified is the rule for choosing the vertex v_i. Many strategies for obtaining this vertex ordering have been proposed. One of the most effective heuristics is the saturation degree ordering (SDO) suggested by Brélaz [2]. The SDO vertex ordering is defined as follows. Suppose that vertices $v_1, \ldots, v_{i-1}$ have been chosen and colored. Vertex v_i is chosen to be a vertex adjacent to the maximum number of different colors in the vertex set $\{v_1, \ldots, v_{i-1}\}$. Note that this heuristic can be implemented to run in time proportional to $\sum_{v \in V} \deg^2(v)$, where $\deg(v)$ is the degree of vertex v.

A modification of the SDO heuristic is the *incidence degree ordering* (IDO) introduced by Coleman and Moré in their work [3] on using coloring heuristics to obtain consistent partitions for sparse Jacobian estimation. The IDO vertex ordering has the advantage that its running time is a linear function of the number of edges. To describe the IDO heuristic, we again suppose that vertices $v_1, \ldots, v_{i-1}$ have been chosen. Vertex v_i is chosen to be a vertex whose degree is a maximum in the subgraph of G induced by the vertex set $\{v_1, \ldots, v_{i-1}\} \cup \{v_i\}$. This heuristic can be implemented to run in time proportional to $\sum_{v \in V} \deg(v)$, or $O(|E|)$ time.

Unfortunately, these heuristics are essentially breadth-first searches of the graph and, as such, do not represent scalable, parallel heuristics. To do better, one notes that if the vertices v and w are not adjacent in G (i.e., are independent in G), then these vertices can be colored in parallel. Thus, a heuristic for determining an independent set in parallel could be adapted to a parallel coloring heuristic. This observation motivates the parallel coloring heuristic shown in Fig. 2.

```
$V' \leftarrow V$;
**While** $V' \neq \emptyset$ **do**
        Choose an independent set $I$ from $V'$;
        Color $I$ in parallel;
        $V' \leftarrow V' \setminus I$;
**enddo**
```

FIG. 2. *Outline of a parallel coloring heuristic.*

The problem of determining an independent set in parallel has been the focus of much theoretical research. An approach that has been successfully analyzed is to determine an independent set, I, based on the following Monte Carlo rule. Here we denote the set of vertices adjacent to vertex v by $\mathrm{adj}(v)$.

1. For each vertex $v \in V'$ determine a distinct, random number $\rho(v)$.
2. $v \in I \Leftrightarrow \rho(v) > \rho(w)$, for all $w \in \mathrm{adj}(v)$.

In the Monte Carlo algorithm described by Luby [8], this initial independent set is augmented to obtain a maximal independent set. The approach is the following. After the initial independent set is found, the set of vertices adjacent to a vertex in I, the neighbor set $N(I)$, is determined. The union of these two sets is deleted from V', the subgraph induced by this smaller set is constructed, and the Monte Carlo step is used to choose an augmenting independent set. This process is repeated until the candidate vertex set is empty and a maximal independent set (MIS) is obtained. The complete Monte Carlo algorithm suggested by Luby for generating an MIS is shown in Fig. 3. In this figure, we denote by $G(V')$ the subgraph of G induced by the vertex set V'. Luby shows that an upper bound for the expected time to compute an MIS by this algorithm on a CRCW P-RAM is $EO(\log(n))$. The algorithm can be adapted to a graph coloring heuristic by using it to determine a sequence of distinct maximal independent sets and by coloring each MIS a different color. Thus, this approach will solve the $(\Delta + 1)$ vertex coloring problem, where Δ is the maximum degree of G, in expected time $EO((\Delta + 1)\log(n))$.

```
$I \leftarrow \emptyset$;
$V' \leftarrow V$;
$G' \leftarrow G$;
**While** $G' \neq \emptyset$ **do**
        Choose an independent set $I'$ in $G'$;
        $I \leftarrow I \cup I'$;
        $X \leftarrow I' \cup N(I')$;
        $V' \leftarrow V' \setminus X$;
        $G' \leftarrow G(V')$;
**enddo**
```

FIG. 3. *Luby's Monte Carlo algorithm for determining a maximal independent set.*

A major deficiency of this approach on currently available parallel computers is that each new choice of random numbers in the MIS algorithm requires a global synchronization of the processors. A second problem is that each new choice of random numbers incurs a great deal of computational overhead, because the data structures associated with the random numbers must be recomputed. In Fig. 4, we present an asynchronous heuristic that avoids both of these drawbacks. The heuristic is written assuming that each vertex v is assigned to a different processor and the processors communicate by passing messages.

```
Choose $\rho(v)$;
$n\text{-}wait = 0$;
$send\text{-}queue = \emptyset$;
For each $w \in \mathrm{adj}(v)$ do
    Send $\rho(v)$ to processor responsible for $w$;
    Receive $\rho(w)$;
    if $(\rho(w) > \rho(v))$ then $n\text{-}wait = n\text{-}wait + 1$;
    else $send\text{-}queue \leftarrow send\text{-}queue \cup \{w\}$;
enddo
$n\text{-}recv = 0$;
While $(n\text{-}recv < n\text{-}wait)$ do
    Receive $\sigma(w)$;
    $n\text{-}recv = n\text{-}recv + 1$;
enddo
$\sigma(v)$ = smallest available color consistent with the
    previously colored neighbors of $v$;
For each $w \in send\text{-}queue$ do
    Send $\sigma(v)$ to processor responsible for $w$;
enddo
```

FIG. 4. *An asynchronous parallel coloring heuristic.*

With the asynchronous heuristic the first drawback (global synchronization) is eliminated by choosing the independent random numbers only at the start of the heuristic. With this modification, the interprocessor communication can proceed asynchronously once these numbers are determined. The second drawback (computational overhead) is alleviated because with this heuristic, once a processor knows the values of the random numbers of the vertices to which it is adjacent, the number of messages it needs to wait for can be computed and stored. Likewise, each processor computes only once the processors to which it needs to send a message once its vertex is colored. Finally, note that this heuristic has more of the "flavor" of the sequential heuristic, since we choose the smallest color consistent with the adjacent vertices previously colored.

One should be concerned that the expected running time of this heuristic is comparable to the expected running time of the heuristic based on the MIS Monte Carlo algorithm. In the next section, we show that under the P-RAM computational model, one is able to obtain an improvement over the upper bound for the expected running time of the MIS algorithm.

3. Expected running time of the asynchronous heuristic.

In this section we derive an upper bound for the expected running time of the heuristic presented in Fig. 4 under the P-RAM computational model. Most of the practical problems we are concerned with are generated from local, physical models. Thus, the maximum degree of the associated graphs is independent of the size of the system. It is reasonable, therefore, to consider the model problem of a graph G with n vertices and bounded degree Δ. Since we assume that Δ is bounded, the previous work by Luby [8] shows that a random heuristic for the $(\Delta+1)$ vertex coloring problem can be accomplished on the P-RAM model in $EO(\log(n))$ time. As discussed above, this heuristic is based on a synchronous random algorithm for determining a sequence of maximal independent sets.

To analyze the running time of our heuristic, we make the following observations. First, for the sake of comparison, we note that this heuristic can also be considered synchronous. Second, we assume that each vertex $v \in V$ chooses a unique independent random number $\rho(v)$. Define a *monotonic path* of length t to be a path of t vertices $\{v_1, v_2, \ldots, v_t\}$ in G, such that $\rho(v_1) > \rho(v_2) > \cdots > \rho(v_t)$. We have the following lemma.

LEMMA 3.1. *The running time of the P-RAM version of the asynchronous heuristic is proportional to the maximum length monotonic path in G.*

Proof. We assume that the asynchronous heuristic can be made synchronous by including a global synchronization point in the receive and send loops (i.e., all processors that have messages to send, send them; all available messages are received; and then the processors synchronize). Since each vertex has degree at most Δ, the number of messages to be received and sent by each processor is bounded by this constant. Under the P-RAM computational model we can assume that messages are sent between processors in constant time. Thus, the running time of the heuristic is proportional to the number of these synchronized steps.

It is clear that the number of steps is at least as long as the longest monotonic path in G. If the number of steps were longer, there would exist some step where the random number of a processor was greater than all its neighbors, yet for some reason it did not send its messages. Hence, we have that the running time of the heuristic is proportional to the maximum length monotonic path. □

To analyze the expected running time of the heuristic, we need to construct an upper bound to the probability of the existence of a monotonic path. We construct this bound in two parts, first by finding a bound on the number of paths of length t, and then by determining the probability that a path is monotonic. The following lemma gives a bound on the number of paths of length t in the graph G.

LEMMA 3.2. *The number, $\eta(t)$, of different paths in G of length t is bounded by*

$$\eta(t) \leq n\Delta(\Delta-1)^{t-2} \quad . \tag{3.1}$$

Proof. This bound can be obtained by induction. For $t=2$, the number of paths is at most $n\Delta$. Now suppose the lemma holds for paths of length $t-1$. Consider some vertex v and all paths of length $t-1$ starting from this vertex. Each of these paths ends at some vertex w. Because an extension of this path cannot return along the edge it used to get to w, there are at most $\Delta - 1$ ways to extend this path. Multiplying the bound for the number of paths of length $t-1$ by $\Delta-1$ yields the desired result. □

Let X be the random variable equal to the maximum length monotonic path in G. Since the random numbers assigned to the vertices are independent, the probability that a path of length t is monotonic is $(1/t!)$. Let $P\{X=t\}$ be the probability that the maximum length monotonic path is t. This probability is bounded by the probability that there exists a monotonic path of length t. Including the bound given in Lemma 3.2, we find

$$P\{X=t\} \leq \frac{\eta(t)}{t!} \leq \frac{n\Delta(\Delta-1)^{t-2}}{t!}. \tag{3.2}$$

THEOREM 3.3. *The expected value of the maximum length monotonic path, EX, is bounded by*

$$EX \le T + (\Delta - 1)^K \tag{3.3}$$

for any K, where T is the minimum integer satisfying

$$T! \ge n\Delta(\Delta - 1)^{T-K-1} \exp(\Delta - 1). \tag{3.4}$$

Proof. The expected value of X is given by

$$EX = \sum_{t=2}^{n} tP\{X = t\}. \tag{3.5}$$

For any integer $T \ge 2$ we have that

$$EX \le T + \sum_{t=T+1}^{n} tP\{X = t\}. \tag{3.6}$$

Thus, we can include the bound on the probability given in (3.2) to obtain

$$\begin{aligned} EX &\le T + \sum_{t=T+1}^{\infty} t\frac{n\Delta(\Delta-1)^{t-2}}{t!} \\ &\le T + \frac{n\Delta}{(\Delta-1)}\sum_{t=T}^{\infty}\frac{(\Delta-1)^t}{t!} \\ &\le T + \frac{n\Delta}{(\Delta-1)}\frac{(\Delta-1)^T}{T!}\exp(\Delta-1)\ . \end{aligned} \tag{3.7}$$

If we choose T to be the minimum integer such that

$$T! \ge n\Delta(\Delta - 1)^{T-K-1} \exp(\Delta - 1)\ , \tag{3.8}$$

we have that the expected maximum length monotonic path is bounded by

$$EX \le T + (\Delta - 1)^K. \tag{3.9}$$ □

As a corollary, we are able to achieve a bound in terms of n for the expected running time of this algorithm. To prove this corollary, we require the following short lemma.

LEMMA 3.4. *The inequality*

$$\frac{r!}{s^r} \ge \sqrt{2\pi s}\left(\frac{\Gamma(r/s)}{e^{1/12}\sqrt{2\pi}}\right)^s \tag{3.10}$$

holds for $r \ge s \ge 1$.

Proof. First, we recall the Gamma function identity

$$r! = r\,\Gamma(r) \tag{3.11}$$

and the formula

$$\Gamma(x) = \sqrt{2\pi}\, x^{x-\frac{1}{2}}\, e^{-x}\, e^{\frac{\theta(x)}{12}}, \tag{3.12}$$

which holds for $x \geq 1$ and for $0 < \theta(x) < 1$. For $r \geq 1$, we compute the lower bound

$$\Gamma(r) \geq \sqrt{\frac{2\pi}{r}} \left(\frac{r}{e}\right)^r \tag{3.13}$$

by setting $\theta = 0$ in (3.12). Thus, we have the bound

$$\frac{r!}{s^r} \geq \sqrt{2\pi r} \left(\frac{r}{se}\right)^r . \tag{3.14}$$

We set $\theta = 1$ in (3.12) to obtain an upper bound for the Gamma function and assume that $r \geq s \geq 1$. We let $r' = r/s$ and find

$$\begin{aligned} \sqrt{2\pi r} \left(\frac{r}{se}\right)^r &= \sqrt{2\pi r} \left(\left(\frac{r'}{e}\right)^{r'} \right)^s \\ &= \sqrt{2\pi r} \left(e^{\frac{1}{12}} \sqrt{\frac{2\pi}{r'}} \left(\frac{r'}{e}\right)^{r'} e^{-\frac{1}{12}} \sqrt{\frac{r'}{2\pi}} \right)^s \\ &\geq \sqrt{2\pi r} \left(\Gamma(r') e^{-\frac{1}{12}} \sqrt{\frac{r'}{2\pi}} \right)^s \\ &\geq \sqrt{2\pi s} \left(\Gamma(r') \frac{e^{-\frac{1}{12}}}{\sqrt{2\pi}} \right)^s . \end{aligned} \tag{3.15}$$

Combining (3.14) and (3.15), we obtain the desired bound

$$\frac{r!}{s^r} \geq \sqrt{2\pi s} \left(\frac{\Gamma(r/s)}{e^{\frac{1}{12}} \sqrt{2\pi}} \right)^s . \tag{3.16}$$

□

The Gamma function is not monotonic for $x \geq 1$. However, it is monotonic for slightly larger x, for example, $x \geq \frac{3}{2}$. To avoid this lack of uniqueness, we define the function $\Gamma^+(y) = \max\{ x \mid \Gamma(x) = y \}$, which is well defined for $y \geq 1$. We now prove the following corollary to Theorem 3.3.

COROLLARY 3.5. *For $\Delta \geq 2$, the expected number of steps*, EX, *for the random heuristic is bounded by*

$$EX \leq (\Delta - 1)\Gamma^+ \left(\sqrt{2\pi}\, e^{\frac{13}{12}} \left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}} \right)^{\frac{1}{\Delta-1}} \right) + 2. \tag{3.17}$$

Proof. Choosing $K = 0$ in Theorem 3.3 and subtracting one from T in equation (3.4), we have the following inequality:

$$\frac{(T-1)!}{(\Delta-1)^{T-1}} \leq \frac{n\Delta}{(\Delta-1)} \exp(\Delta - 1) . \tag{3.18}$$

We assume that $(T-1) \geq (\Delta - 1) \geq 1$, and by Lemma 3.4 we have

$$\sqrt{2\pi(\Delta-1)} \left(\frac{e^{-\frac{1}{12}}}{\sqrt{2\pi}} \Gamma\left(\frac{T-1}{\Delta-1}\right) \right)^{\Delta-1} \leq \frac{(T-1)!}{(\Delta-1)^{T-1}} . \tag{3.19}$$

Combining the bounds in (3.18) and (3.19), we obtain

$$\Gamma\left(\frac{T-1}{\Delta-1}\right) \le \sqrt{2\pi}e^{\frac{13}{12}}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}. \tag{3.20}$$

Using the asymptotic inverse to the Gamma function defined above, we obtain the bound

$$T-1 \le (\Delta-1)\,\Gamma^{+}\left(\sqrt{2\pi}e^{\frac{13}{12}}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}\right). \tag{3.21}$$

Because we have chosen $K = 0$, by Theorem 3.3 we have that $EX \le T + 1$. Thus, consistent with our original assumption that $T \ge \Delta$, we have the desired bound for EX,

$$EX \le (\Delta-1)\,\Gamma^{+}\left(\sqrt{2\pi}e^{13/12}\left(\frac{n\Delta}{\sqrt{2\pi}\,(\Delta-1)^{\frac{3}{2}}}\right)^{\frac{1}{\Delta-1}}\right)+2. \qquad \square \tag{3.22}$$

By using the lower bound for the Gamma function obtained by choosing $\theta(x) = 0$ in (3.12) we note that, for fixed Δ, this bound is asymptotically $EO(\log(n)/\log\log(n))$. This bound is an improvement over the $EO(\log(n))$ upper bound obtained by Luby [8].

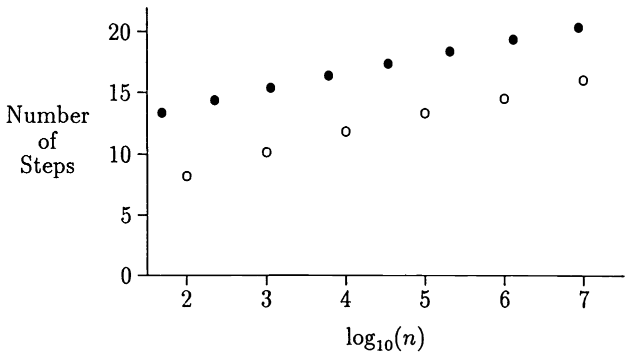


FIG. 5. *Bound obtained by choosing $K = -1$ for the expected value (●) versus experimental average (○) for regular, $\Delta = 4$, graphs.*

The bound given in Theorem 3.3 yields a surprisingly close fit to what we have observed in practice. In Fig. 5, we compare the bound for EX with our experimental results for regular graphs of degree 4. In the plot the open circles are the observed average number of steps for the asynchronous heuristic as a function of the base 10 logarithm of n. The closed circles are obtained from the bound given in Theorem 3.3, where we have chosen $K = -1$. The points are obtained by choosing a value for T and then solving (3.4) for the largest n that satisfies the inequality.

As a final note, we emphasize that, although the heuristics described above have a random component, their behavior in practice is essentially deterministic. In the above analysis, note that the probability that there exists a monotonic path of length greater than t asymptotically decays faster than exponentially. Thus, the bounds on the expected running time hold with very high probability. In addition, Luby [8] gives a prescription for converting his Monte Carlo MIS algorithm into a deterministic algorithm with the same running time. Hence, these heuristics are fundamentally different from those based on simulated annealing. Although the simulated annealing algorithms can be shown to ultimately obtain optimal solutions, running time bounds comparable to those above do not exist.

4. A medium grain heuristic for distributed memory computers.

Our primary interest is the development of a heuristic suitable for distributed memory computers. In this section, we describe how the asynchronous Monte Carlo heuristic presented in the preceding section can be combined with the heuristics that have been successful on sequential machines. Using this approach, in the next section we experimentally demonstrate that for certain classes of problems the performance is scalable.

Consider a distributed memory computer with p processors. We assume that the vertices of the graph $G = (V, E)$ are partitioned across these processors by the sets $\{V_1, \dots, V_p\}$. Let the function $\pi : V \rightarrow \{1, \dots, p\}$ return the number of the partition, or processor, to which each vertex is assigned. We define the *edge separator* E^S to be the set of edges $E^S \subseteq E$ where the edge $(v, w) \in E^S \Leftrightarrow \pi(v) \neq \pi(w)$. In addition, we define the set of *global vertices* to be the vertex set V^S, where a vertex is in this set if and only if the vertex is an endpoint for some edge in E^S. Let the set of *local vertices* V^L be the set $V \setminus V^S$. Finally, denote by V_i^S and V_i^L the vertex sets $V^S \cap V_i$ and $V^L \cap V_i$.

The following theorem shows that it is possible to decompose the asynchronous heuristic into two parts, the first part to color the global vertices, and the second part to color the local vertices. We show that the vertex labeling obtained by piecing together these colorings is a coloring for G. In this theorem we denote the subgraph of G induced by the vertex set V_i by $G(V_i)$.

THEOREM 4.1. *Let σ_S be a coloring for $G(V^S)$. This coloring, restricted to V_i^S, can be independently extended to a coloring σ_i for the subgraph $G(V_i)$. If we define the function σ by $\sigma(v) = \sigma_i(v)$, where $v \in V_i$, then σ is a coloring for G.*

Proof. Consider the vertices V_i on processor i. We assume that vertices V_i^S are consistently colored when the random heuristic colors $G(V^S)$. Thus, only the vertices V_i^L remain to be colored on this processor. By definition, the vertices V_i^L can be connected only to vertices in V_i. Because V_i^S has been colored, the vertices V_i^L may be colored independently from any other vertices in V. By the same observation, we note that if the coloring chosen for each V_i^L is consistent for $G(V_i)$, then these colorings combine to form a consistent coloring for the entire graph. $\square$

From Theorem 4.1, we observe that the parallel graph coloring problem can be accomplished in two phases:

1. Color $G(V^S)$ using the asynchronous Monte Carlo heuristic.
2. On processor i, color $G(V_i^L)$ given $\sigma_S(V_i^S)$ using a sequential heuristic.

A subtle point is that we need the Monte Carlo algorithm to generate independent sets in the graph $G_S = (V^S, E^S)$, not the graph $G(V^S)$. Note that G_S is a sparser graph than $G(V^S)$, since G_S does not contain edges (v, w) where $v, w \in V^S$ but $\pi(v) = \pi(w)$. Such edges are included in $G(V^S)$. We use the notation $\Delta_S = \Delta(G_S)$ and $n_S = |V^S|$.

```
Determine $V_i^S$, $V_i^L$;     {Partition vertices}
$color\text{-}queue = \emptyset$;
For each $v \in V_i^S$ do     {Set up queues for separator vertices}
    $n\text{-}wait(v) = 0$;
    $send\text{-}queue(v) = \emptyset$;
    For each edge $(v,w) \in E^S$ do
        Compute $\rho(w)$;
        if $(\rho(w) > \rho(v))$ then $n\text{-}wait(v) = n\text{-}wait(v) + 1$;
        else $send\text{-}queue(v) \leftarrow send\text{-}queue(v) \cup \{w\}$;
    enddo
    if $(n\text{-}wait(v) = 0)$ then
        $color\text{-}queue \leftarrow color\text{-}queue \cup \{v\}$;
enddo
Seq-color ($\sigma$, $color\text{-}queue$);     {Color any vertices in $V_i^S$ not}
$n\text{-}colored = |\, color\text{-}queue \,|$;     {waiting for messages}
Pack-and-send ($\sigma$, $color\text{-}queue$, $send\text{-}queue$);
$color\text{-}queue = \emptyset$;
While $(n\text{-}colored < |V_i^S|)$ do
    Receive $msg$;
    For each $w \in msg.vertex\text{-}list$ do
        $\sigma(w) = msg.vertex\text{-}color$;
        For each $v \in msg.vertex\text{-}adj$ do
            $n\text{-}wait(v) = n\text{-}wait(v) - 1$;
            if $(n\text{-}wait(v) = 0)$ then
                $color\text{-}queue \leftarrow color\text{-}queue \cup \{v\}$;
        enddo
    enddo
    Seq-color ($\sigma$, $color\text{-}queue$);     {Color subsets of $V_i^S$ once required}
    $n\text{-}colored = n\text{-}colored + |\, color\text{-}queue \,|$;     {messages are received}
    Pack-and-send ($\sigma$,$color\text{-}queue$, $send\text{-}queue$);
    $color\text{-}queue = \emptyset$;
enddo
Seq-color ($\sigma$, $V_i^L$ );     {Color local vertices last}
```

FIG. 6. *A distributed memory parallel coloring heuristic for the ith processor.*

Thus, we have $\Delta_S \leq \Delta(G(V^S))$, and the bounds detailed in Theorem 3.3 depend on the values of Δ_S and n_S.

In Fig. 6 we outline the complete distributed heuristic to be executed by the ith processor. The heuristic calls two procedures: *Seq-color* and *Pack-and-send.* Given a partial coloring of vertices stored in the array σ and a list *queue* of local vertices to be colored, *Seq-color*(σ, *queue*) colors these vertices with a sequential heuristic (such as the IDO heuristic discussed earlier). The procedure *Pack-and-send* sends the vertices in *queue* and their colors σ to nonlocal, adjacent vertices on other processors with lower random numbers. For vertex v this set is stored in the array *send-queue*(v), which is initialized at the beginning of the heuristic.

The ability to pack vertex information into messages allows for the optimization of interprocessor communication. For example, messages sent between processors can be packed to overcome the high message startup cost on machines like the Intel iPSC/860. The data structure *msg* that a processor receives contains a packed list of vertices, their colors, and the vertices assigned to the receiving processor to which they are adjacent. As before, the number of nonlocal vertices that must be colored before vertex v is computed at the beginning of the heuristic and stored in *n-wait*(v).

One last optimization to note is that if the pseudo-random number generator used to determine $\rho(v)$ depends only on the vertex number, then these values do not need to be sent between processors. Instead, each processor can determine these values locally, and the overhead involved with this interprocessor communication can be avoided. This optimization is included in the initialization section of Fig. 6.

5. Experimental results. We have implemented the heuristic described in Fig. 6 in the C programming language on a 64-node Intel iPSC/860. In this section we present results obtained with this implementation. One of our main objectives is to demonstrate the scalability of this heuristic consistent with the definition given in [5]. Thus, we would like to show that, for a fixed number of vertices per processor, the running time of the heuristic is only a slowly increasing function of the number of processors used.[1]

To achieve this objective, we have chosen test problems whose sizes can be easily scaled and that are also representative of problems encountered in applications. The problems we consider are generated from finite element models of structures and from finite difference schemes for two- and three-dimensional regular domains.

We consider two sets of structures problems; both are modeled by using three-dimensional, hexagonal linear elements, where the nonzero structure of the resulting assembled sparse system is used as a test matrix. The problems in Problem Set I are obtained from a model of a long rectangular beam of varying lengths, seven-by-seven finite elements thick, with the degrees of freedom constrained at both ends. The problems in the Problem Set II are generated from a model of a multistoried building of varying heights with constraints applied by elimination of the bottom layer of vertices.

For these two problem sets the vertex to processor assignment was made by assigning to each processor contiguously numbered blocks of columns based on an initial numbering. These blocks consist of n/p columns, where n is the order of the matrix and p is the number of processors. The initial numbering of the columns is chosen such that nearby nodes in the finite element models are generally close in number. Thus, this matrix partition scheme roughly corresponds to a physical partition.

In Tables 1 and 2 we show the sizes of the problems contained in these two sets. The number of vertices in the graphs is listed in the column labeled n, the number of edges is listed under m, and the maximum degree of the graph is shown under Δ. The number of colors used by a *sequential* implementation of the IDO heuristic and the SDO heuristic is given in the columns labeled χ_{IDO} and χ_{SDO}.

We also consider two sets of problems arising from standard finite differencing schemes for regular domains. In Table 3 we show the sizes of the test problems generated for the nine-point stencil on a square, two-dimensional domain. For these problems the domain is partioned into subsquares of equal size, resulting in equal numbers of vertices being assigned to each processor. To keep the aspect ratio of the subsquares the same as the problem is scaled, we change the size of the problems by a factor of four as the problem is scaled. In Table 4 we show the sizes of the test problems generated for the 27-point stencil on a cubic, three-dimensional domain.

[1]In our case, the running time will increase with problem size according to the *slowly* growing function given in Theorem 3.3.

TABLE 1
Problem Set I.

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
CUBE1	1,701	46,623	72	21	18
CUBE2	3,888	113,304	72	20	18
CUBE4	8,262	246,666	72	20	19
CUBE8	17,010	513,390	72	21	19
CUBE16	34,506	1,046,838	72	21	19
CUBE32	69,498	2,113,734	72	21	19
CUBE64	139,482	4,247,526	72	21	19

TABLE 2
Problem Set II.

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
SKY1	6,270	145,554	75	19	19
SKY2	12,540	298,751	76	21	20
SKY4	25,080	605,145	76	21	21
SKY8	50,160	1,217,933	76	21	21
SKY16	100,320	2,443,509	76	21	21
SKY32	200,640	4,894,661	76	22	22

Again, equal numbers of vertices are assigned to each processor because the domain is partioned into subcubes of equal size. For these problems the problem size increases by a factor of eight as the problem is scaled.

TABLE 3
Problem Set III.

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
9PT1	2,500	19,404	8	5	4
9PT4	10,000	78,804	8	5	4
9PT16	40,000	317,604	8	5	4
9PT64	160,000	1,275,204	8	5	4

Scaling results obtained for Problem Sets I–IV are shown in Tables 5–8. For the results presented in these four tables, the partitioning ensures that the average number of vertices per processor, $\langle n \rangle$, is essentially constant. The number of processors used is listed in the column labeled p. The number of vertices and maximum degree of G_S are given under n_S and Δ_S, respectively. The maximum time in seconds used by a processor in coloring G_S is given under T_S. T_L is the maximum time in seconds used by a processor to solve its local coloring problem. The average number of messages sent by the processors is listed under $\langle N_{\text{msg}} \rangle$. Also shown are $\tilde{\chi}_S$, the number of colors used in coloring G_S, and $\tilde{\chi}$, the number of colors used to color the entire graph. For these results the IDO heuristic is used to solve the local coloring problems.

Note that although we used the incidence degree heuristic to solve the local coloring problem for G_L, for the results presented in Tables 5–8, one could also employ the more expensive saturation degree heuristic. Recall that the SDO heuristic requires the colors used to color adjacent vertices to compute the saturation degree of each vertex. This information has already been communicated to each processor prior to the coloring of G_L; therefore, the SDO heuristic can be used to color G_L without necessitating any additional interprocessor communication. In Tables 9–12, we present the results for the parallel heuristic modified in this manner.

TABLE 4
Problem Set IV.

Problem	n	m	Δ	χ_{IDO}	χ_{SDO}
27PT1	2,197	48,456	26	12	11
27PT8	17,576	421,400	26	12	12
27PT64	140,608	3,511,656	26	13	12

TABLE 5
Parallel coloring results for Problem Set I, IDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
CUBE1	1	1701	0	0	0.000	0.330	0.0	0	21
CUBE2	2	1944	486	27	0.076	0.368	6.5	14	21
CUBE4	4	2091	2,136	66	0.273	0.379	20.2	24	26
CUBE8	8	2126	5,436	66	0.541	0.376	28.6	24	25
CUBE16	16	2157	11,975	69	0.531	0.375	36.1	25	26
CUBE32	32	2172	25,004	70	0.488	0.378	37.8	27	27
CUBE64	64	2179	51,031	70	0.588	0.381	39.0	26	26

TABLE 6
Parallel coloring results for Problem Set II, IDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
SKY1	2	3,135	1,518	59	0.226	0.417	31.5	21	23
SKY2	4	3,135	3,624	65	0.408	0.408	118.2	23	24
SKY4	8	3,135	7,836	65	0.572	0.420	152.2	22	24
SKY8	16	3,135	16,260	65	0.584	0.412	166.8	23	25
SKY16	32	3,135	33,108	65	0.582	0.410	174.1	24	25
SKY32	64	3,135	66,804	65	0.583	0.408	177.4	25	26

TABLE 7
Parallel coloring results for Problem Set III, IDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
9PT1	1	2,500	0	0	0.000	0.084	0.0	0	5
9PT4	4	2,500	396	5	0.015	0.089	3.2	4	7
9PT16	16	2,500	2,364	5	0.017	0.090	5.1	5	7
9PT64	64	2,500	11,004	5	0.028	0.089	5.1	5	7

TABLE 8
Parallel coloring results for Problem Set IV, IDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
27PT1	1	2,197	0	0	0.000	0.183	0.0	0	12
27PT8	8	2,197	3,752	19	0.124	0.179	82.4	14	15
27PT64	64	2,197	43,272	19	0.270	0.176	179.9	15	17

TABLE 9
Parallel coloring results for Problem Set I, SDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
CUBE1	1	1701	0	0	0.000	6.167	0.0	0	18
CUBE2	2	1944	486	27	0.077	7.630	6.5	14	18
CUBE4	4	2091	2,136	66	0.274	7.675	20.2	24	25
CUBE8	8	2126	5,436	66	0.550	7.722	28.6	24	25
CUBE16	16	2157	11,975	69	0.536	7.577	36.1	25	25
CUBE32	32	2172	25,004	70	0.498	7.771	37.8	27	27
CUBE64	64	2179	51,031	70	0.593	7.732	39.0	26	26

TABLE 10
Parallel coloring results for Problem Set II, SDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
SKY1	2	3,135	1,518	59	0.233	6.361	31.5	21	22
SKY2	4	3,135	3,624	65	0.415	6.130	118.2	23	24
SKY4	8	3,135	7,836	65	0.578	6.357	152.4	22	23
SKY8	16	3,135	16,260	65	0.584	6.097	166.9	23	24
SKY16	32	3,135	33,108	65	0.585	6.141	174.2	24	25
SKY32	64	3,135	66,804	65	0.588	6.018	177.4	24	25

TABLE 11
Parallel coloring results for Problem Set III, SDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
9PT1	1	2,500	0	0	0.000	0.364	0.0	0	4
9PT4	4	2,500	396	5	0.015	0.407	3.2	4	6
9PT16	16	2,500	2,364	5	0.025	0.405	5.1	5	7
9PT64	64	2,500	11,004	5	0.028	0.403	5.6	5	7

TABLE 12
Parallel coloring results for Problem Set IV, SDO *used to solve local problem.*

Problem	p	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\text{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
27PT1	1	2,197	0	0	0.000	1.666	0.0	0	11
27PT8	8	2,197	3,752	19	0.126	1.540	82.4	14	15
27PT64	64	2,197	43,272	19	0.275	1.618	179.3	15	16

The results shown in Tables 5–12 demonstrate the scalable performance of the heuristic: for a fixed number of nonzeros per processor, the time required by the global and local phases is essentially constant [5]. Note that as the size of G_S increases, the average number of messages sent per processor gradually increases. By maintaining a reasonable average message size, the high communication overhead on the iPSC/860 can be partially amortized. Also, note that by using the SDO heuristic to solve the local coloring problem, a slight improvement in the total number of colors can be obtained. However, the SDO heuristic is significantly more expensive than the IDO heuristic in solving the local coloring problem.

In Tables 13 and 14, we fix the number of processors at 32 and examine the effect on the performance of the heuristic by varying the number of nonzeros per processor. For these results we use the IDO heuristic to solve the local coloring problem.

Overall, the number of colors required is relatively constant, even though the percentage of the vertices in G_S varies dramatically. To some extent this effect can be explained by noting that even though the relative size of G_S is increasing, the local structure of the separators is essentially the same, since the separators arise from physical partitions of a regular domain. In Table 14, when the relative size of G_S does became small enough to allow Δ_S to decrease, the number of colors used to color G_S, $\tilde{\chi}_S$, decreased. Finally, we note the good performance of the heuristic, both in terms of the number of colors used and execution time, as the size of the local problems becomes quite small.

6. Concluding remarks.

We have presented a new parallel graph coloring heuristic well suited to distributed memory computers. Experimental results demonstrate that this heuristic is scalable and that it produces colorings usually requiring no more than three or four more colors than the best-known linear time sequential

TABLE 13
Parallel coloring results for Problem Set I, $p = 32$.

Problem	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\mathrm{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
CUBE1	53	1,701	71	0.061	0.003	6.4	24	26
CUBE2	122	3,888	70	0.127	0.009	16.1	25	27
CUBE4	258	8,217	70	0.202	0.020	32.8	25	28
CUBE8	532	15,561	70	0.309	0.056	56.5	26	29
CUBE16	1,078	28,842	69	0.385	0.138	82.5	28	30
CUBE32	2,172	25,004	70	0.488	0.378	37.8	27	27
CUBE64	4,359	25,166	70	0.573	0.853	38.7	25	27

TABLE 14
Parallel coloring results for Problem Set II, $p = 32$.

Problem	$\langle n \rangle$	n_S	Δ_S	T_S	T_L	$\langle N_{\mathrm{msg}} \rangle$	$\tilde{\chi}_S$	$\tilde{\chi}$
SKY1	196	6,270	64	0.128	0.011	23.0	23	25
SKY2	392	11,313	65	0.212	0.043	45.7	25	26
SKY4	784	18,012	65	0.396	0.081	74.3	24	25
SKY8	1,568	22,716	65	0.432	0.199	104.4	24	25
SKY16	3,135	33,108	65	0.582	0.410	174.1	24	25
SKY32	6,270	28,272	24	0.491	0.988	106.3	21	24

heuristics. We have also shown that under the P-RAM computational model, this heuristic has an expected run time bounded by $EO(\log(n)/\log\log(n))$.

This parallel heuristic takes full advantage of locality in the generation of the graph. For example, if the graph is generated by the assembly of a structures model or obtained from the spatial decomposition of a physical model, the asynchronous random phase of the heuristic can efficiently color the global separator. After the separator is colored, the remaining problem decomposes into independent local coloring problems. The only constraint on these local colorings is that they be consistent with the coloring determined for the separator. Thus, any sequential heuristic can be used to solve each of these local coloring problems simultaneously.

For many problems a physical partition can be used to generate a good vertex to processor assignment. When the determination of a partition is not straightforward, a partitioning heuristic would have to be used. For example, recent advances in the automatic partitioning of three-dimensional domains [11] or in spectral dissection methods [9] could be employed. We note that a partitioning that maintains locality is advantageous, although not essential, to the performance of the parallel heuristic. The heuristic requires only that the number of vertices assigned per processor allow for good load balancing.

An interesting observation is that even if the coloring obtained for the separator uses more colors than a good sequential heuristic, the separator subgraph is usually sparser than the entire graph. Thus, when coloring the denser local subgraphs, some of the difference between the parallel and sequential heuristics in the number of colors used for the separator subgraph can be offset by the use of a good sequential heuristic to color the remaining local subgraphs.

Finally, motivated by the following observation, we note a possible avenue for improving the heuristic. When one observes the distribution of colors produced by the heuristic, one often sees very few vertices using the highest colors. For example, when coloring the graph 27PT8 on eight processors, the results in Table 12 show that 15 colors were required by the parallel algorithm, but only 12 by the sequential algorithm. However, the number of vertices using the colors 15, 14, and 13 were

2, 12, and 70, respectively. An interesting topic for further research might be the introduction of a postprocessing step that would attempt to recolor these few vertices with lower color values, and thus decrease the total number of colors used.

Acknowledgment. We thank Tim Kiemel for help in the probabilistic analysis of the expected maximum length monotone path, and John Gilbert for bringing the paper by Michael Luby to our attention. We also thank the referees for detailed and helpful comments.

REFERENCES

[1] B. Bollobás, *Graph Theory*, Springer-Verlag, New York, 1979.

[2] D. Brélaz, *New methods to color the vertices of a graph*, Comm. ACM, 22 (1979), pp. 251–256.

[3] T. F. Coleman and J. J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, New York, 1979.

[5] J. L. Gustafson, G. R. Montry, and R. E. Benner, *Development of parallel methods for a 1024-processor hypercube*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 609–638.

[6] D. S. Johnson, *Worst case behavior of graph coloring algorithms*, in Proc. 5th Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica Publishing, Winnipeg, 1974, pp. 513–527.

[7] M. T. Jones and P. E. Plassmann, *Scalable iterative solution of sparse linear systems*, Preprint MCS-P277-1191, Mathematics and Computer Science Div., Argonne National Laboratory, Argonne, IL, 1991.

[8] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 4 (1986), pp. 1036–1053.

[9] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal., 11 (1990), pp. 430–452.

[10] R. Schreiber and W.-P. Tang, *Vectorizing the conjugate gradient method*, Unpublished manuscript, Dept. of Computer Science, Stanford Univ., Stanford, CA, 1982.

[11] S. Vavasis, *Automatic domain partitioning in three dimensions*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 950–970.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 670–676, May 1993

A CHOLESKY UP- AND DOWNDATING ALGORITHM FOR SYSTOLIC AND SIMD ARCHITECTURES*

CHRISTIAN H. BISCHOF[†], C.-T. PAN[‡], AND PING TAK PETER TANG[†]

Abstract. This paper presents an algorithm for maintaining Cholesky factors of symmetric positive definite matrices under arbitrary rank-one changes. The algorithm synthesizes Carlson's updating algorithm, and the downdating algorithm recently suggested by Pan to arrive at an algorithm which is both simple and allows for the pipelining of up- and downdates (in any order). On an array of $O(n^2)$ processors, the algorithm allows an $n \times n$ matrix to be updated at a cost per update that is independent of n. Implementation results on the 1024-processor AMT DAP-510 emphasize the simplicity and practicality of the proposed scheme.

Key words. Cholesky factorization, updating, downdating, systolic algorithm, SIMD algorithm

AMS(MOS) subject classifications. 15A23, 65F05, 68Q10

1. Introduction.

Given a symmetric positive definite matrix A, an important factorization of A is the Cholesky factorization

$$A = R^T R$$

where R is upper triangular. In many applications, it is desirable to recalculate the Cholesky factorization after A has been changed by a rank-one modification

$$\tilde{A} = A \pm zz^T = R^T R \pm zz^T.$$

Calculating the new factorization for $A + zz^T$ is known as *the updating problem*; recomputing $A - zz^T$ is called *the downdating problem* (see [5] for an overview).

Our work was mainly motivated by the recursive least squares filtering problem [1]. In that setting, one has to rapidly calculate a sequence of Cholesky factors $R^{(k)}$ of rank-one modified matrices

$$(R^{(k)})^T R^{(k)} = R^T R + \sigma_1 z_1 z_1^T + \cdots \sigma_k z_k z_k^T, \qquad \sigma_j \in \{-1, 1\}, k = 1, 2, \ldots.$$

Typically, k is much larger than n, the dimension of R, and often the $R^{(k)}$'s must be calculated in real time. Hence we would like to design a systolic algorithm where we maintain the Cholesky factor $R^{(k)}$ using $O(n^2)$ processors, while update vectors z_j are streaming through the processor array. To minimize the time per update, we would further like to overlap the processing of different updates so that processing of a new update vector can start at every time step. By pipelining the processing of updates in this fashion, the computation of $R^{(k)}$ is completed in $O(n + k)$ time steps, so that on the average a rank-one modification requires constant time.

The standard updating algorithm in LINPACK and the less well known one by Carlson [4] can easily be pipelined when a sequence of updates occurs. These algorithms are based on Givens rotations and operate on the upper triangular R in one pass from top to bottom.

*Received by the editors August 6, 1990; accepted for publication (in revised form) March 6, 1992. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under contract W-31-109-Eng-38. Computations were performed at Argonne's Advanced Research Computing Facility.

[†]Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439 (`bischof@mcs.anl.gov` and `tang@mcs.anl.gov`).

[‡]Department of Mathematical Sciences, Northern Illinois University, DeKalb, Illinois 60115 (`pan@math.niu.edu`). This work was performed while the author was visiting Argonne National Laboratory under Argonne's Faculty Research Participation program.

Unfortunately, this is not the case in the LINPACK downdating algorithm originally due to Saunders [10]. That algorithm completes the downdating task in two passes: a triangular solve that processes R from top to bottom, and then a set of Givens rotations that process R from bottom to top.

Recently, Pan proposed a variant of the LINPACK algorithm [8] that is at least as stable as the one in LINPACK but more economical and operates R only once—from top to bottom. In this paper, we show that by suitably combining Pan's downdating algorithm with Carlson's updating algorithm, we arrive at an algorithm that

- employs only orthogonal transformations, thereby avoiding the potential stability problems associated with hyperbolic transformations [1],
- allows the pipelining of any sequence of rank-one modifications, and
- is simple and well suited for systolic arrays and SIMD machines.

The simplicity of the algorithm, together with its low complexity, is its main virtue: the computation is extremely regular, data flow is straightforward, and no "special cases" arise. As a result, an efficient implementation of the algorithm on systolic arrays and SIMD machines is an easy task which does not require tricky coding.

We also mention that in most applications, and in particular in recursive least squares, the update of the Cholesky factor is usually followed by an equation solve. Pan and Plemmons [9] suggested a scheme to update the inverse of the Cholesky factor directly. A MIMD implementation of this scheme is described in [6]. This so-called "co-variance approach" is attractive since it replaces the triangular solve by a matrix-vector multiply. We chose here to consider the update of the Cholesky factor (the so-called "square root approach"), since it avoids potential stability problems with ill-conditioned factors, and because of the availability of efficient algorithms to handle the triangular solves on SIMD machines [11], [12].

In the next section, we present the up- and downdating algorithm that computes arbitrary sequences of rank-one modifications in a pipelined fashion. In §3 we consider the implementation of our algorithm on a massively parallel SIMD machine. Because our algorithm is simple, we are able to derive a very short and efficient program implementing our algorithm. Experiments on the 1024-processor Active Memory Technology DAP-510 parallel computer confirm the practicality and efficiency of our approach. Lastly, we summarize our results.

2. A pipelined algorithm for arbitrary up- and downdates.

Let

$$R = \begin{pmatrix} r_1^T \\ r_2^T \\ \vdots \\ r_n^T \end{pmatrix}$$

be the Cholesky factor of an $n \times n$ symmetric positive definite matrix, where r_j^T is the jth row of R. Furthermore, let

$$z^T = (\zeta_1, \zeta_2, \dots, \zeta_n)$$

be the vector determining the rank-one update or downdate.

Carlson's algorithm [4] for calculating T, $T^T T = R^T R + zz^T$, and Pan's algorithm [8] for calculating T, $T^T T = R^T R - zz^T$, are summarized in Figs. 1 and 2.

If one juxtaposes the algorithms in this fashion, it becomes apparent that there is a lot of commonality between them:

- the calculations for the α_j's and $z^{(j)}$'s are identical,

```
$\beta_0 := 1;\ z^{(0)} := z$
For $j = 1, 2, \ldots, n$ do
  $\alpha_j$   :=  $\zeta_j^{(j-1)}/r_{jj}$
  $\beta_j$   :=  $\sqrt{\beta_{j-1}^2 + \alpha_j^2}$
  $z^{(j)}$  :=  $z^{(j-1)} - \alpha_j r_j$
  $t_j$   :=  $(\beta_{j-1} r_j + \alpha_j z^{(j-1)}/\beta_{j-1})/\beta_j$
end do
```

FIG. 1. *Carlson's updating algorithm.*

```
$\beta_0 := 1;\ z^{(0)} := z$
For $j = 1, 2, \ldots, n$ do
  $\alpha_j$   :=  $\zeta_j^{(j-1)}/r_{jj}$
  $\beta_j$   :=  $\sqrt{\beta_{j-1}^2 - \alpha_j^2}$
  $z^{(j)}$  :=  $z^{(j-1)} - \alpha_j r_j$
  $t_j$   :=  $(\beta_j r_j - \alpha_j z^{(j)}/\beta_j) \Big/ \beta_{j-1}$
end do
```

FIG. 2. *Pan's downdating algorithm.*

- the calculations for the β_j's differ by only a sign, and
- the calculations for t_j share the factor $\sigma\alpha_j/(\beta_{j-1}\beta_j)$, where $\sigma = 1$ for update and $\sigma = -1$ for downdate.

As a result, one can easily merge the two algorithms to arrive at the algorithm shown in Fig. 3. Again the triangular factor R is overwritten with the updated factor T. When $\sigma = 1$, it is identical to Carlson's algorithm; when $\sigma = -1$, it is identical to Pan's algorithm.

```
$\beta_0 := 1;\ z^{(0)} := z$
For $j = 1, 2, \ldots, n$ do
  $\alpha_j := \zeta_j^{(j-1)}/r_{jj}$
  $\beta_j := \sqrt{\beta_{j-1}^2 + \sigma\alpha_j^2}$
  $z^{(j)} := z^{(j-1)} - \alpha_j r_j$
  if($\sigma == -1$) then { **downdate** }
    $t_j := (\beta_j/\beta_{j-1}) r_j + \sigma(\alpha_j/(\beta_j\beta_{j-1})) z^{(j)}$
  elseif ($\sigma == +1$) then {**update** }
    $t_j := (\beta_{j-1}/\beta_j) r_j + \sigma(\alpha_j/(\beta_{j-1}\beta_j)) z^{(j-1)}$
  end if
end do
```

FIG. 3. *Algorithm for up- and downdating.*

Recall that our objective is to calculate the Cholesky factorization of

$$(R^{(k)})^T R^{(k)} = R^T R + \sigma_1 z_1 z_1^T + \cdots \sigma_k z_k z_k^T, \qquad \sigma_j \in \{-1, 1\}, k = 1, 2, \ldots$$

for $k \gg n$ in $n + k$ steps. To that end we have to overlap the processing of different updates so that processing of different updates can start at every time step. Specifically, we wish to achieve the situation depicted in Fig. 4. Let us assume that initially

$$\begin{pmatrix} r & r & r & r & r \\ 0 & r & r & r & r \\ 0 & 0 & r & r & r \\ 0 & 0 & 0 & r & r \\ 0 & 0 & 0 & 0 & r \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & r & r & r & r \\ 0 & 0 & r & r & r \\ 0 & 0 & 0 & r & r \\ 0 & 0 & 0 & 0 & r \end{pmatrix} \Longrightarrow \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & r & r & r \\ 0 & 0 & 0 & r & r \\ 0 & 0 & 0 & 0 & r \end{pmatrix}$$

$$\Longrightarrow \begin{pmatrix} 3 & 3 & 3 & 3 & 3 \\ 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & r & r \\ 0 & 0 & 0 & 0 & r \end{pmatrix} \Longrightarrow \begin{pmatrix} 4 & 4 & 4 & 4 & 4 \\ 0 & 3 & 3 & 3 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & r \end{pmatrix} \Longrightarrow \begin{pmatrix} 5 & 5 & 5 & 5 & 5 \\ 0 & 4 & 4 & 4 & 4 \\ 0 & 0 & 3 & 3 & 3 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

FIG. 4. *Pipelining up- and downdates.*

$R = R^{(0)}$ is stored in the processor array, as is indicated by the "r" entries. In the first time step, we update the first row of R to that of $R^{(1)}$ as shown in the second matrix of Fig. 4. Here (as in the following snapshots) a row of $R^{(j)}$ is denoted by j's. computed. In the next time step, we then compute the first row of $R^{(2)}$ and the second row of $R^{(1)}$ and so on. Thus, after k steps, the processor array would contain (from top to bottom)

$$(1) \qquad \begin{pmatrix} (r_1^{(k)})^T \\ (r_2^{(k-1)})^T \\ \vdots \\ (r_{n-1}^{(k-n)})^T \\ (r_n^{(k-n+1)})^T \end{pmatrix},$$

where

$$R^{(j)} = \begin{pmatrix} (r_1^{(j)})^T \\ \vdots \\ (r_n^{(j)})^T \end{pmatrix}$$

is the Cholesky factor after j up- and downdates have been performed.

Since our algorithm processes R in one pass from top to bottom, it is amenable to a pipelining implementation along the lines of Fig. 4. If we have $O(n^2)$ processors, we can dedicate j processors to computing the jth row of the updated R for $j = 1, \ldots, n$. As a result, the update of an r_j stored in a processor row takes constant time, and a new update vector z can be processed at every time step. We mention that this algorithm does compare favorably with other approaches to up- and downdating, in particular those based on hyperbolic rotations (see, for example, [2], [3] , [6], [8]; these articles contain further pointers on the rather substantial literature on this subject).

3. Experimental results. To demonstrate the practicality of our algorithm, we implemented it on an Active Memory Technology 1024-processor DAP-510 computer at Argonne's Advanced Computing Research Facility.

The DAP consists of a 32×32 array of one-bit processors and operates in SIMD mode driven by a 10-MHz clock (for an architecture overview, see [7]). The 32-bit arithmetic (applied to 1024 numbers at the same time) requires on the average 850 cycles for an add, 920 cycles for a square root, 1400 cycles for a multiply, and 1950 cycles for a divide. Communication primitives are very efficient. For example, a "spread" operation that turns a vector

$$x = \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_n \end{pmatrix}$$

into a matrix

$$X = \begin{pmatrix} \xi_1 & \cdots & \xi_1 \\ \vdots & & \vdots \\ \xi_n & \cdots & \xi_n \end{pmatrix}$$

requires only 170 cycles. The DAP is not a powerful machine in terms of Mflops, but since arithmetic is performed in a bit-sliced fashion, an implementation can easily be custom tailored to specific bit lengths. That capability, together with its small power and space requirements, makes it an interesting machine for real-time embedded signal-processing applications.

The DAP can be conveniently programmed in DAP-Fortran, which contains extensions to Fortran that allow one to express operations on whole matrices and vectors with ease. Apart from the "spread" operation, we also use the "diag" primitive to extract the diagonal of a matrix in a vector, and the "shift_down" primitive, which returns

$$\begin{pmatrix} 0 \\ \xi_1 \\ \vdots \\ \xi_{n-1} \end{pmatrix}$$

for a vector x and works in an analogous fashion for matrices. Because of the SIMD nature of the machine, operations on matrices and vectors are to be understood elementwise, in other words, $C = A * B$ implies $c_{ij} = a_{ij} * b_{ij}$. Operations can be restricted to certain elements in a matrix or vector by using a so-called mask, which is an array of logical values that is **true** where an operation is to be applied. For two masks, mask1 and mask2, say, that partition a given array, we use the $\cup$ notation if a matrix is composed out of disjoint pieces, e.g.,

$$C = A(mask1) \cup B(mask2),$$

to stress the fact that no arithmetic is performed. In the description of the algorithm we use the "not" function to generate such complementary partitions.

It is easy to translate the algorithm in Fig. 3 into an efficient program for an SIMD machine such as the DAP. The program is shown in Fig. 5; to be consistent with the previous notation, we refer to matrices by upper-case roman letters, vectors by lower-case roman letters, and scalars by Greek letters. Built-in functions are denoted by bold type. The array R contains the updated Cholesky factor in the staggered form as in (1); Z contains the updated vectors $z^{(j)}$; s contains the σ's and b the β's; and *mask_down* is a mask indicating which rows are being updated and which are being downdated.

```
$Z = 0$; $s = 0$; $b = 1$; $mask\_down =$ **false**;
**for** $j = 1, 2, \ldots$ **do**
$Z(1, 1:n) = z_j$; $s(1) = \sigma_j$; $b(1) = 1$;
    $mask\_down(1) = \begin{cases} \textbf{true}, if \sigma_j = -1; \\ \textbf{false}, otherwise. \end{cases}$
    $b\_previous = b$; $Z\_previous = Z$;
    $a = \textbf{diag}(Z)/\textbf{diag}(R)$;
    $b = \textbf{sqrt}(b\_previous * *2 + s * (a * *2))$;
    $Z = Z\_previous - \textbf{spread}(a) * R$;
    $a\_bar = s * a/(b * b\_previous)$;
    $b\_bar = (b(mask\_down) \cup b\_previous(\textbf{not}(mask\_down)))$
            $/(b\_previous(mask\_down) \cup b(\textbf{not}(mask\_down)))$;
    $Z\_bar = Z\_previous(\textbf{spread}(\textbf{not}(mask\_down))) \cup Z(\textbf{spread}(mask\_down))$;
    $R = \textbf{spread}(b\_bar) * R + \textbf{spread}(a\_bar) * Z\_bar$;
    **shift\_down**($a$,$b$,$s$,$mask\_down$); **shift\_down**($Z$);
**end do**
```

FIG. 5. SIMD *program for computing rank-one Cholesky updates in a pipelined fashion.*

The simplicity of the algorithm is crucial to allow for such a short program. Particularly beneficial is the fact that we have only one if-statement whose branches perform the same operations (on different data). Remember that on an SIMD machine the branches of an if-statement

If (c) then a else b endif

are executed sequentially. First, all processors evaluate c, then the processors for which c evaluated to "true" evaluate a, and then the processors for which c evaluated to "false" evaluate b. If a and b contain arithmetic (which is expensive), the program is slowed down considerably. In our setting, statements a and b contain only data moves (which are much faster than arithmetic), since the update of R requires the same arithmetic operations (two multiplies and an add). Thus, if we ignore the time for data moves, one time step in our algorithm requires one square root, three adds, three divides, and eight multiplications. It is also easy to see that in one pass through the loop we compute (in parallel) $\frac{3}{2}n^2 + 5n$ multiplies, $n^2 + n$ adds, $3n$ divisions, and n square roots, and hence our SIMD algorithm does not compute any redundant floating-point operations compared to the sequential version.

The execution time of some experiments with matrices of size 32, 64, and 128 (computed with 32-bit floating-point arithmetic) is shown in Fig. 6. Here we generated random vectors z_j and alternated between up- and downdates. The straight lines confirm that the time per update is indeed constant: It is 2.9 milliseconds for 32×32 matrices, 5.9 milliseconds for 64×64 matrices, and 16.9 milliseconds for 128×128 matrices. Since we are limited to a fixed-size processor array (instead of having $O(n^2)$ as we assumed so far), the execution time does increase with problem size. Once n is greater than 32, matrices are folded onto the smaller 32×32 processor array, so one would expect the execution time to increase with $(n/32)^2$. In reality we do not see such growth, since we are using the hardware more efficiently when working with larger matrices: For 32×32 matrices we are only using half the available processors, for a 128×128 matrix we have 16 32×32 tiles, only four of which are triangular; all others are full.

4. Conclusions. Building on the work of Carlson [4] and Pan [8], we developed a stable algorithm for maintaining Cholesky factors of symmetric positive definite

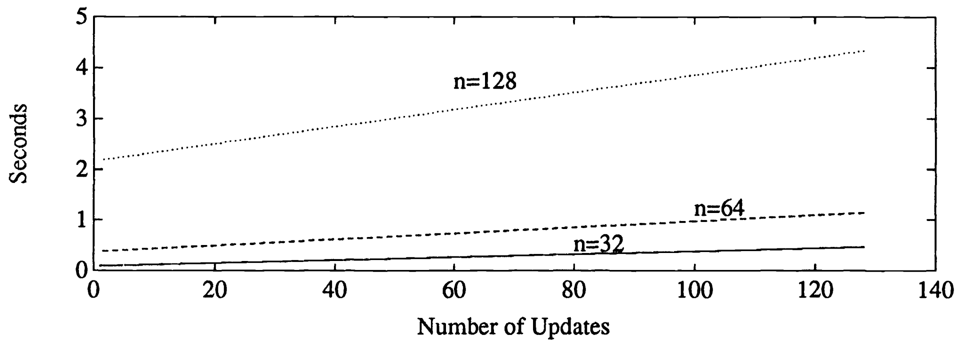


FIG. 6. *Execution time as a function of the number of updates performed.*

matrices under arbitrary rank-one modifications (as long as they maintain positive definiteness). Each rank-one modification requires only $\frac{3}{2}n^2$ multiplications and n^2 additions; and, for the most part, the computation of up- or downdates involves the same operation. As a result, one can easily pipeline different rank-one updates to arrive at an algorithm that on a systolic array or massively parallel SIMD machine requires constant time per update.

The new algorithm is well suited for signal processing applications. Its computational complexity compares favorably to that of other approaches, it avoids the potential stability problems associated with hyperbolic rotations, and as our implementation on the 1024-processor AMT DAP has shown, an efficient implementation on an SIMD machine or systolic array can be easily obtained.

REFERENCES

[1] S. ALEXANDER, C. PAN, AND R. PLEMMONS, *Analysis of a recursive least squares hyperbolic rotation algorithm for signal processing*, Linear Algebra Appl., 98 (1988), pp. 3–40.

[2] C. H. BISCHOF, C.-T. PAN, AND P. T. P. TANG, *Stable Cholesky up- and downdating algorithms for systolic and* SIMD *architectures*, Tech. Rep. MCS-P167-0790, Mathematics and Computer Science Div., Argonne National Laboratory, Argonne, IL 1990.

[3] A. W. BOJANCZYK, R. P. BRENT, P. V. DOOREN, AND F. R. D. HOOG, *A note on downdating the Cholesky factorization*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 210–221.

[4] N. A. CARLSON, *Fast triangular factorization of the square root filter*, AIAA Journal, 11 (1973), pp. 1259–1265.

[5] P. GILL, G. GOLUB, W. MURRAY, AND M. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[6] C. S. HENKEL AND R. J. PLEMMONS, *Recursive least squares computations on a hypercube multiprocessor using covariance factorizations*, SIAM J. Sci. Statist. Comput., 1 (1991), pp. 95–106.

[7] R. M. HORD, *Parallel Supercomputing in* SIMD *Architectures*, CRC Press, 1990.

[8] C.-T. PAN, *A modification of the LINPACK downdating algorithm*, BIT, (1990), pp. 707–722.

[9] C.-T. PAN AND R. J. PLEMMONS, *Least squares modifications with inverse factorizations: Parallel implications*, J. Comput. Appl. Math., 27 (1989), pp. 109–127.

[10] M. SAUNDERS, *Large-scale linear programming using the Cholesky factorization*, Tech. Rep. STAN-CS-72-252, Dept. of Computer Science, Stanford Univ., Palo Alto, CA, 1972.

[11] R. SCHREIBER AND P. J. KUEKES, *Systolic linear algebra machines in digital signal processing*, in VLSI and Modern Signal Processing, S. V. Kung, H. J. Whitehouse, and T. Kailath, eds., 1985, Prentice-Hall, Englewood Cliffs, NJ, pp. 389–405.

[12] R. SCHREIBER AND W.-P. TANG, *On systolic arrays for updating Cholesky factorizations*, BIT, 26 (1986), pp. 451–466.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 677–692, May 1993

INDEX REDUCTION IN DIFFERENTIAL-ALGEBRAIC EQUATIONS USING DUMMY DERIVATIVES*

SVEN ERIK MATTSSON† AND GUSTAF SÖDERLIND‡

Abstract. A new index reduction algorithm for DAEs is developed. In the usual manner, parts of the DAE are differentiated analytically and appended to the original system. For each additional equation, a derivative is selected to be replaced by a new algebraic variable called a *dummy derivative.* The resulting augmented system is at most index 1, but no longer overdetermined. The dummy derivatives are not subject to discretization; their purpose is to annihilate part of the dynamics in the DAE, leaving only what corresponds to the dynamics of a state-space form. No constraint stabilization is necessary in the subsequent numerical treatment. Numerical tests indicate that the method yields results with an accuracy comparable to that obtained for the corresponding state-space ODE.

Key words. differential-algebraic, index reduction, dummy derivative, constraint stabilization, solution invariant, automatic differentiation

AMS(MOS) subject classification. 65L05

1. Introduction.

We shall develop a new index reduction technique for differential-algebraic equations (DAEs)

$$F(t,x,\dot{x}) = 0. \tag{1.1}$$

We assume that the problem is *solvable* [4], with a unique, smooth solution when supplied with an appropriate number of consistent initial conditions. The *index* of the problem equals the minimum number of times that all or part of (1.1) must be differentiated with respect to t in order to determine $\dot{x}$ as a continuous function of x and t [4] and is assumed to be constant along the solution. In addition, F is assumed to be sufficiently differentiable to allow the proposed index reduction technique. For alternative definitions of the index, cf. [11] or [14].

It is well known that it is numerically difficult to solve a high-index DAE. Index reduction methods provided in the literature (see, e.g., [4, p. 33]) can be used as a remedy. However, it is often less satisfactory to solve the underlying ordinary differential equation, or UODE, that has been derived from the DAE through index reduction. The reason is that the set of solutions to the UODE is larger than the corresponding set of solutions to the original DAE; the algebraic relations of the DAE are only implicit in the UODE as solution invariants. Unless linear, these invariants are generally not preserved under discretization. As a result, the numerical solution drifts off the algebraic constraints, often leading to instabilities [10]. Consequently, so-called constraint stabilization techniques have been devised [1], [3], [12].

To avoid such difficulties, one may try to obtain a low-index formulation, with a solution set identical to that of the original problem. This can be achieved by augmenting the system as the index reduction proceeds: all original equations and their successive derivatives are retained in the process. The result is an overdetermined but consistent index-1 DAE. Like invariants, however, consistency is generally lost

*Received by the editors July 10, 1991; accepted for publication (in revised form) May 15, 1992. This research was supported in part by Swedish Research Council for Engineering Sciences contract 222/91-405.

†Department of Automatic Control, Lund Institute of Technology, Box 118, S-221 00 Lund, Sweden.

‡Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden.

when the system is discretized. Therefore, special projection techniques are required for the numerical solution (see, e.g., [8] or [9]). Related approaches using least-squares projections have been used in [2], [5], and [6].

The index reduction technique proposed in this paper, which was outlined in [16], overcomes the latter complication by introducing a new dependent variable for each new equation generated in the reduction process. This generates an augmented, determined index-1 DAE from the original problem. The method is related to classical reduction to state-space form; the remaining dynamics of the problem is represented by a selection of its original variables, forming a (local) UODE. The other variables are solved for in terms of the UODE states. The technique is applicable to large classes of problems and can be practically implemented, using symbolic or automatic differentiation [13], [18].

We shall outline our method in a simple example. Consider the DAE problem

$$\dot{x} = y, \tag{1.2a}$$

$$\dot{y} = z, \tag{1.2b}$$

$$x = f(t). \tag{1.2c}$$

This is an index-3 *derivative chain* with solution $x = f(t)$, $y = \dot{f}(t)$, and $z = \ddot{f}(t)$. It may be thought of as prototypical for prescribed-trajectory problems in mechanics, where one wants to calculate the (usually generalized) forces required for the system to accomplish the desired action. In such an application x, y, and z would represent position, velocity, and force per unit mass, respectively, and $f(t)$ is the prescribed trajectory for the system.

A sufficient condition for the index to be at most 1 is that we are able to solve for the highest-order derivatives in the system, i.e., $\dot{x}$, $\dot{y}$, and z. This is obviously impossible in (1.2). Differentiating (1.2a) once and (1.2c) twice yields, after reordering equations and variables,

$$\ddot{x} = \ddot{f}(t), \tag{1.2c$''$}$$

$$\dot{y} = \ddot{x}, \tag{1.2a$'$}$$

$$z = \dot{y}, \tag{1.2b}$$

which is index 1. Here (1.2a′) denotes the derivative with respect to t of (1.2a).

Now, consider the overdetermined but consistent system obtained by augmenting the original system by the successive derivatives of (1.2a) and (1.2c):

$$x = f(t), \tag{1.3c}$$

$$\dot{x} = \dot{f}(t), \tag{1.3c$'$}$$

$$\ddot{x} = \ddot{f}(t), \tag{1.3c$''$}$$

$$y = \dot{x}, \tag{1.3a}$$

$$\dot{y} = \ddot{x}, \tag{1.3a$'$}$$

$$z = \dot{y}. \tag{1.3b}$$

For each *differentiated equation* appended to the original system, we need one "new" dependent variable to make the augmented system determined rather than overdetermined. This is achieved by replacing one derivative from each *differentiated* equation by a new algebraic variable. Thus we eliminate $\ddot{x}$ by substituting a *dummy derivative* x'' for $\ddot{x}$ wherever it occurs in the system (1.3). Although $x'' \equiv \ddot{x}$, *the dummy* *derivative is a purely algebraic variable and is not subject to discretization.* Similarly, we replace $\dot{x}$ by x' and $\dot{y}$ by y'. We have chosen this notation for the dummy derivatives to directly indicate what they represent and how they were introduced into the augmented system.

We have thus obtained the augmented but determined system

$$x = f(t), \tag{1.4a}$$

$$x' = \dot{f}(t), \tag{1.4b}$$

$$x'' = \ddot{f}(t), \tag{1.4c}$$

$$y = x', \tag{1.4d}$$

$$y' = x'', \tag{1.4e}$$

$$z = y'. \tag{1.4f}$$

This purely algebraic (hence index-1) system is mathematically equivalent to (1.2). No initial conditions can be imposed, and no discretization is required for its numerical solution. The system is nonsingular since it is possible to solve algebraically for the six unknowns x, x', x'', y, y', and z. Although this example is rather special, it demonstrates an important aspect of the new reduction technique: that the system can be solved numerically without discretizing all derivatives.

In the following sections we shall describe how to proceed in the general case. We first show how to obtain the appropriate differentiated system and then how to select dummy derivatives to make the augmented system determined. We shall also deal with pivoting of the selected set of dummy derivatives, before presenting numerical results.

2. Differentiation and permutation of equations.

For notational convenience, we shall rewrite (1.1) as an operator equation

$$\mathcal{F}x = 0. \tag{2.1}$$

The dependent variables may appear algebraically or differentiated up to q times. We assume that for some $p \geq q$ the $\mathbb{R}^n$-valued function $x \in \mathcal{C}^p$. Similarly, $\mathcal{F}x \in \mathcal{C}^{p-q}$ is an $\mathbb{R}^n$-valued function.

Let $D = d/dt$ denote the differentiation operator, and let $\nu \in \mathbb{N}^n$ denote a multi-index $\nu = (\nu_1, \nu_2, \ldots, \nu_n)^T$. Then we define $D^\nu = \operatorname{diag}(D^{\nu_1}, \ldots, D^{\nu_n})$. We let $\mu(\mathcal{F}) \in \mathbb{N}^n$ denote a multi-index such that $D^{\mu(\mathcal{F})}x$ are the highest-order derivatives appearing in the DAE, i.e., $x_j^{(\mu_j)}$ is the highest-order derivative of x_j that appears in $\mathcal{F}x = 0$.

2.1. Structural properties and permutations.

Following [4, p. 21], we call a property of a matrix a *generic* or *structural property* if it holds almost everywhere in a neighborhood of the particular values of the *nonzero entries* of the matrix. A matrix A is structurally nonsingular if and only if there exists a permutation P_1 such that P_1A has a nonzero diagonal, often referred to as a *maximum transversal* or an *output set*. A structurally singular matrix is also singular, but the converse is not true. Similar definitions for a nonlinear system $g(v) = 0$ are obtained by requiring that the system's structural Jacobian can be permuted to obtain a nonzero diagonal. Likewise, we call the DAE problem (1.1) structurally nonsingular if there is an output set when we consider x to be unknown and do not distinguish algebraic and differentiated appearances of x.

We shall make use of block lower triangular (BLT) partitioning in order to decompose a problem into subproblems. By means of a simultaneous row and column permutation of P_1A, the matrix is transformed into $Q^T P_1 A Q = PAQ$, a BLT matrix with the same nonzero diagonal elements as P_1A. Output assignment and BLT partitioning are standard techniques in sparse matrix analysis; cf. [7].

2.2. Differentiations. For a structurally nonsingular DAE $\mathcal{F}x = 0$, it is always possible to find a differentiated problem $\mathcal{G}x = D^\nu \mathcal{F}x = 0$ with ν finite, such that the differentiated problem is structurally nonsingular with respect to its highest-order derivatives $D^{\mu(\mathcal{G})}x$. Pantelides's algorithm (cf. [17]) intended for finding consistent initial values for a DAE, establishes the minimum number of times each equation has to be differentiated, i.e., it finds the minimal $\nu(\mathcal{F})$. His algorithm also constructs the output set of interest automatically, and in the structural analysis step we need only construct the desired BLT partition.

By definition, the problem $\mathcal{F}x = 0$ is index 0 if it uniquely determines the highest-order derivatives $D^{\mu(\mathcal{F})}x$, with all $\mu_j(\mathcal{F}) > 0$, as continuous functions of t and lower derivatives. If the same condition holds with some $\mu_j(\mathcal{F}) = 0$, it is Index 1. The fact that it is not possible to solve uniquely for the highest-order derivatives does not imply that the index is greater than 1: consider the simple index-1 problem:

$$\dot{x} + \dot{y} = 1, \qquad x - y = 0.$$

Thus Pantelides's algorithm may call for an unnecessary differentiation step (here it would suggest differentiating the second equation once, resulting in an index-0 system) but it is simple to remove such superfluous differentiations. Whether this should be done or not depends on if one prefers to be able to solve for the highest-order derivatives or to reduce the index to 1 while introducing as few new variables as possible.

In our algorithm below, we must be able to solve for the highest-order derivatives. We defer comments on the singular case to §4.

2.3. Differentiation, permutation, index reduction algorithm. The index reduction procedure for $\mathcal{F}x = 0$ consists of the following steps:

1. If the problem is structurally singular, then return an error message.
2. *Differentiation.* Use Pantelides's algorithm to obtain
   (a) a vector $\nu(\mathcal{F}) \in \mathbb{N}^n$,
   (b) a differentiated problem $\mathcal{G}x = \mathcal{F}^\nu x = 0$ with $\mathcal{F}^\nu = D^{\nu(\mathcal{F})}\mathcal{F}$,
   (c) an output set for $\mathcal{F}^\nu x = 0$ with respect to its highest-order derivatives $D^{\mu(\mathcal{G})}x$ as unknowns.
3. *Permutation.* Since $P\mathcal{F}^\nu Q = PD^{\nu(\mathcal{F})}P^T P\mathcal{F}Q = D^{P\nu(\mathcal{F})}P\mathcal{F}Q$, BLT partitioning with respect to unknowns $D^{\mu(\mathcal{G})}x$ yields the permuted
   (a) undifferentiated problem $\mathcal{H}y = 0$ with $\mathcal{H} = P\mathcal{F}Q$ and $y = Q^T x$,
   (b) differentiated problem $\mathcal{H}^{P\nu}y = 0$ with $\mathcal{H}^{P\nu} = P\mathcal{F}^\nu Q$, i.e., $\nu(\mathcal{H}) = P\nu(\mathcal{F})$, and the latter problem is BLT with respect to its highest-order derivatives $D^{Q^T\mu(\mathcal{G})}y$.
4. *Index reduction.* Select derivatives to be replaced by dummy derivatives, blockwise as indicated by the BLT partition. If unable to select one dummy for each differentiated equation (singular case), manipulate original equations using information from differentiated equations and restart at step 1.

In the rest of this paper we will focus on the central index reduction step, i.e., how to select dummy derivatives. Note (cf. Step 4 below) that this relies on finding a *nonsingular* submatrix, implying that this step is numerical/mathematical as opposed to the two structural steps of differentiation and permutation. To simplify the notation we will without loss of generality view the BLT partitioned problem as our original problem, i.e., $\mathcal{F} = \mathcal{H}$.

3. Selection of dummy derivatives.

Consider the differentiated problem $\mathcal{G}x = \mathcal{F}^\nu x = 0$ in BLT form. For notational simplicity, let $g_i = 0$ represent the ith block of $\mathcal{G}x = 0$, let z_i denote the vector of highest-order derivatives of the unknowns associated with that block, and note that $\dim z_i = \dim g_i$. Let $f_i = 0$ be the corresponding block of the original problem $\mathcal{F}x = 0$. We make the following assumptions:

A1. $\mathcal{G}x = \mathcal{F}^\nu x = 0$ is in BLT form.

A2. The equations in each block have been sorted in descending order with respect to number of differentiations, i.e., $\nu_1(g_i) \geq \nu_2(g_i) \geq \cdots$.

A3. The Jacobian $\partial g_i/\partial z_i$ evaluated at the current point on the solution has full rank (the singular case will be discussed later).

To obtain an equivalent index-1 problem, the reduction algorithm constructs a sequence of $\nu_1(g_i) + 1$ problems indexed by a superscript $[j]$.

3.1. Index reduction algorithm.

Step 1: Set $z_i^{[1]} \leftarrow z_i$; $g_i^{[1]}(z_i^{[1]}) \leftarrow g_i(z_i)$; $G_i^{[1]} = \partial g_i^{[1]}/\partial z_i^{[1]} \leftarrow \partial g_i/\partial z_i$; $j \leftarrow 1$.

Step 2: If $g_i^{[j]} = 0$ has no differentiated equations, go to Step 6.

Step 3: If $g_i^{[j]} = 0$ has m differentiated equations, let $h_i^{[j]} = 0$ denote its m first equations. The Jacobian $H_i^{[j]} = \partial h_i^{[j]}/\partial z_i^{[j]}$ then equals the first m rows of $G_i^{[j]}$, due to the sorting with respect to number of differentiations.

Step 4: Next, select m columns $l_1, \ldots, l_m$ of $H_i^{[j]}$ to make a square, nonsingular matrix $M_i^{[j]}$. Selected columns indicate derivatives to be replaced: from the m equations $h_i^{[j]} = 0$ we select the m components of $\hat{z}_i^{[j]} = (\, z_{i,l_1}^{[j]} \ \cdots \ z_{i,l_m}^{[j]} \,)^T$ to be replaced (in Step 6) by dummy derivatives.

Step 5: Now decide how to use the *predecessor* of $h_i^{[j]} = 0$, denoted by $D^{-1}h_i^{[j]} = 0$, thus omitting the last differentiation. New candidates $z_i^{[j+1]}$ for possible replacement are $D^{-1}\hat{z}_i^{[j]}$, i.e., derivatives of one order less than those selected in Step 4. The components of $\hat{z}_i^{[j]}$ are all highest-order derivatives in $h_i^{[j]} = 0$ and are differentiated at least once. Hence, they represent derivatives of the original unknowns x, implying that $D^{-1}\hat{z}_i^{[j]}$ is well defined. The Jacobian $\partial g_i^{[j+1]}/\partial z_i^{[j+1]} = M_i^{[j]}$, as will be shown later. Thus, set

$$g_i^{[j+1]} \leftarrow D^{-1}h_i^{[j]}; \quad z_i^{[j+1]} \leftarrow D^{-1}\hat{z}_i^{[j]}; \quad G_i^{[j+1]} \leftarrow M_i^{[j]}; \quad j \leftarrow j+1,$$

and repeat from Step 2.

Step 6: Let $k_i = j$. We now obtain an index-1 formulation of $f_i = 0$ by collecting all original and all differentiated equations:

$$\begin{pmatrix} g_i^{[k_i]} \\ \vdots \\ g_i^{[1]} \end{pmatrix} = 0.$$

In all equations, introduce a unique *dummy derivative* for each derivative selected in Step 4 (i.e., the variables given by $\hat{z}_i^{[k_i]}, \ldots, \hat{z}_i^{[1]}$), to replace that derivative wherever it occurs. The system now consists of the original equations $f_i = 0$ and the sequence of differentiated equations leading to $g_i = 0$. The unknowns are the original ones and the newly introduced all-algebraic dummy derivatives. □

Before proving that the algorithm converts $f_i = 0$ to an equivalent index-1 formulation, let us first consider its application to a simple linear problem.

EXAMPLE 1. Let our problem $\mathcal{F}x = 0$ be defined as

$$x_1 + x_2 + u_1(t) = 0, \tag{3.1a}$$
$$x_1 + x_2 + x_3 + u_2(t) = 0, \tag{3.1b}$$
$$x_1 + \dot{x}_3 + x_4 + u_3(t) = 0, \tag{3.1c}$$
$$2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + u_4(t) = 0, \tag{3.1d}$$

where the $u_m(t)$ are known forcing functions. The differentiated problem $\mathcal{G}x = 0$ is

$$\ddot{x}_1 + \ddot{x}_2 + \ddot{u}_1(t) = 0, \tag{3.1a$''$}$$
$$\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \ddot{u}_2(t) = 0, \tag{3.1b$''$}$$
$$\dot{x}_1 + \ddot{x}_3 + \dot{x}_4 + \dot{u}_3(t) = 0, \tag{3.1c$'$}$$
$$2\ddot{x}_1 + \ddot{x}_2 + \ddot{x}_3 + \dot{x}_4 + u_4(t) = 0, \tag{3.1d}$$

which was obtained by differentiating (3.1a) and (3.1b) twice and (3.1c) once. The vector of highest-order derivatives in $\mathcal{G}x = 0$ is $z_1 = (\ddot{x}_1 \quad \ddot{x}_2 \quad \ddot{x}_3 \quad \dot{x}_4)^T$.

Block triangularization results in a single block, $g_1(x) = \mathcal{G}x$, and the differentiated problem is index 0, since the Jacobian with respect to the highest-order derivatives

$$\frac{\partial g_1}{\partial z_1} = G_1^{[1]} = \begin{array}{c} \\ (a'') \\ (b'') \\ (c') \\ (d) \end{array} \begin{array}{c} \begin{array}{cccc} \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \dot{x}_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

is nonsingular. This matrix is set up in Step 1. Throughout the steps of the index reduction, we indicate at the Jacobians which equations and variables are presently being considered. The reduction process essentially takes $G_1^{[1]}$ and successively deletes rows and columns. The proper choice of dummy derivatives is deduced during this process.

We have three differentiated equations, and Step 3 gives

$$H_1^{[1]} = \begin{array}{c} \\ (a'') \\ (b'') \\ (c') \end{array} \begin{array}{c} \begin{array}{cccc} \ddot{x}_1 & \ddot{x}_2 & \ddot{x}_3 & \dot{x}_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}.$$

At Step 4 we have two possibilities to select a nonsingular submatrix of $H_1^{[1]}$; columns 1, 3, and 4 or columns 2, 3, and 4. Let us take the first alternative:

$$M_1^{[1]} = \begin{array}{c} \\ (a'') \\ (b'') \\ (c') \end{array} \begin{array}{c} \begin{array}{ccc} \ddot{x}_1 & \ddot{x}_3 & \dot{x}_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \end{array}.$$

Thus we use (3.1a″), (3.1b″), and (3.1c′) to replace the derivatives $\ddot{x}_1$, $\ddot{x}_3$, and $\dot{x}_4$. Then, at Step 5 we prepare for the next cycle and consider the predecessor of the present subproblem, omitting one differentiation:

$$G_1^{[2]} = \begin{array}{c} \\ (\text{a}') \\ (\text{b}') \\ (\text{c}) \end{array} \begin{array}{c} \begin{array}{ccc} \dot{x}_1 & \dot{x}_3 & x_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \end{array}.$$

Now $j = 2$, and we repeat from Step 2. Since we still have differentiated equations, Step 3 yields

$$H_1^{[2]} = \begin{array}{c} \\ (\text{a}') \\ (\text{b}') \end{array} \begin{array}{c} \begin{array}{ccc} \dot{x}_1 & \dot{x}_3 & x_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \end{array}.$$

At Step 4 we have to select the two first columns, and we obtain

$$M_1^{[2]} = \begin{array}{c} \\ (\text{a}') \\ (\text{b}') \end{array} \begin{array}{c} \begin{array}{cc} \dot{x}_1 & \dot{x}_3 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array}.$$

Thus we use (3.1a′) and (3.1b′) to replace $\dot{x}_1$ and $\dot{x}_3$. Step 5 therefore results in

$$G_1^{[3]} = \begin{array}{c} \\ (\text{a}) \\ (\text{b}) \end{array} \begin{array}{c} \begin{array}{cc} x_1 & x_3 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array}.$$

Since there are no differentiated equations, we go to Step 6 and collect the pieces. To get a simple and clear notation, we let x_1'' denote the dummy derivative that is substituted for $\ddot{x}_1$, and similarly for other derivatives.

(3.2a) $$x_1 + x_2 + u_1(t) = 0,$$
(3.2b) $$x_1 + x_2 + x_3 + u_2(t) = 0,$$

(3.2a′) $$x_1' + \dot{x}_2 + \dot{u}_1(t) = 0,$$
(3.2b′) $$x_1' + \dot{x}_2 + x_3' + \dot{u}_2(t) = 0,$$
(3.2c) $$x_1 + x_3' + x_4 + u_3(t) = 0,$$

(3.2a″) $$x_1'' + \ddot{x}_2 + \ddot{u}_1(t) = 0,$$
(3.2b″) $$x_1'' + \ddot{x}_2 + x_3'' + \ddot{u}_2(t) = 0,$$
(3.2c′) $$x_1' + x_3'' + x_4' + \dot{u}_3(t) = 0,$$
(3.2d) $$2x_1'' + \ddot{x}_2 + x_3'' + x_4' + u_4(t) = 0.$$

The nine unknown variables are x_1, x_3, x_1', x_3', x_4, x_1'', x_3'', x_4', and x_2, which are all algebraic except for x_2, which appears differentiated twice.

The problem is now index 1, with the equations ordered as in Step 6. It is BLT with respect to its highest-order derivatives. The first block consists of the equations from the last loop of the selection procedure, which gave (3.2a) and (3.2b), from which we can solve for x_1 and x_3. The second block consists of the equations from the second-to-last loop, which gave (3.2a′), (3.2b′), and (3.2c), from which we can solve x_1', x_3', and x_4. The third and last block consists of the equations from the first loop, which gave (3.2a″), (3.2b″), (3.2c′), and (3.2d), from which we can solve x_1'', $\ddot{x}_2$, x_3'', and x_4'. □

3.2. Proof of index reduction. We shall outline proofs of the crucial parts of Steps 4, 5, and 6.

Step 4: First we will prove by induction that one can select a nonsingular matrix $M_i^{[j]}$ at Step 4. By assumption A3, $G_i^{[1]}$ has full rank. If $G_i^{[j]}$ has full rank and there are differentiated equations, then $H_i^{[j]}$ has full rank. Hence it is possible to select a square nonsingular submatrix $M_i^{[j]}$. Defining $G_i^{[j+1]} = M_i^{[j]}$ then implies that $G_i^{[j+1]}$ has full rank. By induction it follows that it is possible to select a full rank matrix $M_i^{[j]}$ each time we arrive at Step 4.

Step 5: Next, we show the formula $G_i^{[j+1]} = M_i^{[j]}$ at Step 5. It implies that $G_i^{[j]} = \partial g_i^{[j]}/\partial z_i^{[j]}$. Consider the element p, q of $G_i^{[j+1]}$. We have

$$G_{i,pq}^{[j+1]} = \partial g_{i,p}^{[j+1]}/\partial z_{i,q}^{[j+1]} = \partial D^{-1} g_{i,p}^{[j]}/\partial D^{-1} z_{i,l_q}^{[j]},$$

and for the same element of $M_i^{[j]}$, we have

$$M_{i,pq}^{[j]} = \partial h_{i,p}^{[j]}/\partial \hat{z}_{i,q}^{[j]} = \partial g_{i,p}^{[j]}/\partial z_{i,l_q}^{[j]}.$$

The formula $G_i^{[j+1]} = M_i^{[j]}$ now follows directly from the fact that for any differentiable function $e(t, v)$, $\partial e(t, v)/\partial v = \partial \dot{e}(t, v, \dot{v})/\partial \dot{v}$.

Step 6: We shall now show that the problem generated by the algorithm is at most index 1. We first consider the reduction of a block. The equations are $g_i^\star(z_i^\star) = 0$ with

$$g_i^\star = \begin{pmatrix} g_i^{[k_i]} \\ \vdots \\ g_i^{[1]} \end{pmatrix}, \qquad z_i^\star = \begin{pmatrix} z_i^{[k_i]} \\ \vdots \\ z_i^{[1]} \end{pmatrix}.$$

Considered as an algebraic problem, $g_i^\star(z_i^\star) = 0$ is BLT, where we can take the jth block to consist of the equations $g_i^{[j]} = 0$ with the unknowns $z_i^{[j]}$. Since the Jacobian $\partial g_i^{[j]}/\partial z_i^{[j]} = G_i^{[j]}$ is nonsingular, we can solve for all the unknowns $z^\star$. The variables of $z_i^{[j]}$ are of two categories. First, $z_i^{[j]}$ contains the variables $\hat{z}_i^{[j]}$ selected for replacement in Step 5 of the jth loop. The remaining variables clearly represent highest-order derivatives of the old variables x in the problem generated by the algorithm. Thus $z^\star$ contains all the highest-order derivatives of the problem generated by the algorithm, and since we can solve for them, the problem is at most index 1.

Finally, consider the complete problem. Sort the block subsystems $g_i^{[j]} = 0$ with respect to descending order of the index j, and with respect to descending order of the index i. The unknowns $z_i^{[j]} = 0$ are sorted similarly. The resulting problem is then BLT with nonsingular blocks. Consequently, the complete problem is at most index 1. □

The original problem and the problem resulting from the algorithm are mathematically equivalent in the sense that they have identical solution sets in the original (undifferentiated) dependent variables x. Original algebraic equations are still explicitly present. The advantage of the index reduction technique proposed here is that it excludes some derivatives from discretization; by treating the dummy derivatives as algebraic variables, the problem of inconsistency due to discretization is eliminated.

3.3. Removing superfluous differentiations. As was mentioned in §2, Pantelides's algorithm differentiates some problems to index 0 and others to index 1. If the differentiated problem is index 0, the number of dummy derivatives introduced can be decreased by reintroducing some of them as ordinary derivatives. This also reduces the total number of equations accordingly. The procedure is to modify Step 6 in the following way:

1. Do not include the equations of $g_i^{[1]} = 0$, which are differentiated.

2. Do not replace the corresponding selected components $\hat{z}_i^{[1]}$ with dummy derivatives, but replace them with the first-order derivative of the representatives for their predecessors, $D^{-1}\hat{z}_i^{[1]}$, some of which may be dummy derivatives.

EXAMPLE 2. Let us again consider the problem discussed in Example 1. The differentiated problem (3.1) is index 0, since it contains no algebraic variables. This implies that we can obtain a smaller index-1 formulation by disregarding the most differentiated equations, i.e., (3.2a″), (3.2b″), and (3.2c′). The corresponding highest-order derivatives $\ddot{x}_1$, $\ddot{x}_3$, and $\dot{x}_4$ are not replaced by dummy derivatives, but with dx_1'/dt, dx_3'/dt and $\dot{x}_4$, respectively. The problem then becomes

$$x_1 + x_2 + u_1(t) = 0, \tag{3.3a}$$
$$x_1 + x_2 + x_3 + u_2(t) = 0, \tag{3.3b}$$

$$x_1' + \dot{x}_2 + \dot{u}_1(t) = 0, \tag{3.3a'}$$
$$x_1' + \dot{x}_2 + x_3' + \dot{u}_2(t) = 0, \tag{3.3b'}$$
$$x_1 + x_3' + x_4 + u_3(t) = 0, \tag{3.3c}$$

$$2\dot{x}_1' + \ddot{x}_2 + \dot{x}_3' + \dot{x}_4 + u_4(t) = 0. \tag{3.3d}$$

The six unknowns are x_1, x_3, x_1', x_3', x_4, and x_2, which are all dynamic except for the two algebraic variables x_1 and x_3. The problem is index 1, since by differentiating all but the last equation once, we obtain an index-0 problem. □

This technique may also be applied to some problems that Pantelides's algorithm differentiates to index 1, viz., if the differentiated problem contains index-0 subproblems. Such subproblems may then be treated as in Example 2.

4. Pivoting of dummy derivatives. The algorithm described above assumes that the Jacobian with respect to the highest-order derivatives of the differentiated problem $\mathcal{F}^\nu x = 0$ is nonsingular. If the Jacobian is singular, a more detailed analysis is required. It should be noted that the index reduction algorithm given above transforms the original system *locally* to index 1. Clearly, it could happen that the Jacobian becomes singular along the solution trajectory. The singularity may be due to a (locally) inappropriate selection of dummy derivatives. Therefore, this selection must, in general, be dynamic; i.e., we must be prepared to "pivot" the selection of dummy derivatives.

EXAMPLE 3. The inevitable pendulum. Consider a planar pendulum of mass m and length L. In Cartesian coordinates, the equations of motion are

$$x^2 + y^2 - L^2 = 0, \tag{4.1a}$$

(4.1b) $$m\ddot{x} + (\lambda/L)x = 0,$$
(4.1c) $$m\ddot{y} + (\lambda/L)y + mg = 0,$$

where g is the gravitational constant and λ is the force in the string; the index is 3.

Applying Pantelides's algorithm, the length constraint is differentiated twice:

(4.1a) $$x^2 + y^2 - L^2 = 0,$$
(4.1a′) $$2x\dot{x} + 2y\dot{y} = 0,$$
(4.1a″) $$2x\ddot{x} + 2\dot{x}^2 + 2y\ddot{y} + 2\dot{y}^2 = 0.$$

Thus, the differentiated problem is

(4.1a″) $$2x\ddot{x} + 2\dot{x}^2 + 2y\ddot{y} + 2\dot{y}^2 = 0,$$
(4.1b) $$m\ddot{x} + (\lambda/L)x = 0,$$
(4.1c) $$m\ddot{y} + (\lambda/L)y + mg = 0.$$

The Jacobian J of the differentiated problem with respect to the highest-order derivatives $(\ddot{x} \quad \ddot{y} \quad \lambda)^T$ is

$$J = \begin{pmatrix} 2x & 2y & 0 \\ m & 0 & x/L \\ 0 & m & y/L \end{pmatrix}.$$

Since $\det(J) = -2m(x^2+y^2)/L = -2mL$, the differentiated problem is index 1 if and only if $m \neq 0$ and $L \neq 0$. In this case, the index reduction algorithm gives

$$\begin{array}{ccccc} & & \ddot{x} & \ddot{y} & \lambda \\ H_1^{[1]} = & (\mathrm{a}'') & (2x & 2y & 0). \end{array}$$

Neither x nor y is necessarily nonzero for all times. Due to the length constraint, however, they are not simultaneously zero. By choosing $M_1^{[1]} = (2x)$ when $|x| > |y|$ and $M_1^{[1]} = (2y)$ otherwise, we obtain a well-conditioned $M_1^{[1]}$.

Consider first the case $|x| > |y|$. Selecting $F_1^{[1]} = (2x)$ implies that the differentiated length constraint will be used to replace $\ddot{x}$ and $\dot{x}$, and we obtain

(4.2a) $$x^2 + y^2 - L^2 = 0,$$
(4.2a′) $$2xx' + 2y\dot{y} = 0,$$
(4.2a″) $$2xx'' + 2x'^2 + 2y\ddot{y} + 2\dot{y}^2 = 0,$$
(4.2b) $$mx'' + (\lambda/L)x = 0,$$
(4.2c) $$m\ddot{y} + (\lambda/L)y + mg = 0.$$

When $|x| \leq |y|$, we select $F_1^{[1]} = (2y)$. We then obtain the index-1 problem

(4.3a) $$x^2 + y^2 - L^2 = 0,$$
(4.3a′) $$2x\dot{x} + 2yy' = 0,$$
(4.3a″) $$2x\ddot{x} + 2\dot{x}^2 + 2yy'' + 2y'^2 = 0,$$
(4.3b) $$m\ddot{x} + (\lambda/L)x = 0,$$
(4.3c) $$my'' + (\lambda/L)y + mg = 0.$$

Changing the decision as to whether (4.1a″) and (4.1a′) should be used to replace $\ddot{x}$ and $\dot{x}$ or $\ddot{y}$ and $\dot{y}$, can be interpreted as a pivoting operation, and we refer to it at as *dummy pivoting.* From this example, it can be seen that dummy pivoting corresponds to a (locally necessary) change of mathematical models; cf. [15]. Switching from one model to the other is a simple operation, since in the augmented index-1 problem we solve also for the dummy derivatives. The initial values necessary to continue the numerical integration with the new model are readily available, and the need for restarting the integration can usually be avoided. □

A very simple first approach to handle dummy pivoting is to make a selection at the start and use the resulting equations as long as the numerical DAE solver is able to function properly and pivot only when necessary. In general, however, it is preferable to pivot so that the matrices $M_i^{[j]}$ remain well conditioned. It is therefore desirable to have a rather close integration of the index reduction method and the numerical DAE solver. This suggests that index reduction computations should be incorporated into the solver, rather than being considered as a separate preprocessing tool.

EXAMPLE 4. Lagrangian equations of motion. The Lagrangian equations of the first kind for a constrained mechanical system are written as a second-order equation:

$$M(p)\ddot{p} = F(p,\dot{p}) - G^T(p)\lambda, \tag{4.4a}$$
$$0 = g(p), \tag{4.4b}$$

where p is an n-vector representing the system's position, M is the nonsingular mass matrix, F represents applied forces, and λ is the m-vector of Lagrange multipliers associated with the m constraints (4.4b), assumed to have a full-rank constraint matrix $G(p) = \partial g/\partial p$. Thus the system has $n - m$ degrees of freedom, and the system of equations (4.4) is index 3. Two differentiations of the constraint equation results in

$$M(p)\ddot{p} = F(p,\dot{p}) - G^T(p)\lambda, \tag{4.4a}$$
$$0 = g(p), \tag{4.4b}$$
$$0 = G\dot{p}, \tag{4.4b$'$}$$
$$0 = \dot{G}\dot{p} + G\ddot{p}. \tag{4.4b$''$}$$

Here $\dot{G} = G'\dot{p}$, where G' is the Fréchet derivative of G with respect to p. We can solve for the highest-order derivatives $\ddot{p}$ and λ, since the regularity assumptions imply that $GM^{-1}G^T$ is regular. Selecting dummy derivatives implies choosing m dummies among the n variables $\dot{p}$, and m among $\ddot{p}$, using (4.4b′) and (4.4b″). Then we have a total of $n+3m$ equations, $n+m$ original variables p and λ, and $2m$ dummy derivatives (selected elements of $\dot{p}$ and $\ddot{p}$). The application of dummy derivatives to the Euler–Lagrange equations is similar to the method of *generalized coordinate partitioning.*

It is also instructive to apply the index reduction algorithm to the corresponding first-order system

$$\dot{p} = v, \tag{4.5a}$$
$$M(p)\dot{v} = F(p,v) - G^T(p)\lambda, \tag{4.5b}$$
$$0 = g(p), \tag{4.5c}$$

where the n-vector v represents velocity. Applying Pantelides's algorithm now requires also that (4.5a) is differentiated once, and as before, the resulting differentiated problem is second order. Apart from the same dummies as those selected in (4.4), one will have to select n dummy derivatives $v' = \dot{v}$. However, this variable merely accounts for a trivial action of substitution and can immediately be eliminated. If a first-order system is required, it can easily be obtained after a few more similar operations; it is identical to the first-order system that can be obtained from (4.4). $\square$

Let us finally comment on the singular case. Then, if the index of the differentiated problem is greater than one, it is, at least in principle, possible to derive an equation in which none of the unknown highest-order derivatives appears. However, since we need the whole sequence of differentiated equations, we shall manipulate the original, undifferentiated system so that the corresponding differentiated problem has the desired structure with a nonsingular Jacobian.

EXAMPLE 5. A singular system. Consider the following problem [4, p. 23]:

$$\dot{x} + t\dot{y} = f_1, \tag{4.6a}$$

$$x + ty = f_2. \tag{4.6b}$$

The problem is solvable, and the unique solution is $x = f_2 - t(\dot{f}_2 - f_1)$, $y = \dot{f}_2 - f_1$. Pantelides's algorithm differentiates the second equation once and yields

$$\dot{x} + t\dot{y} = f_1, \tag{4.6a}$$

$$\dot{x} + t\dot{y} + y = \dot{f}_2. \tag{4.6b$'$}$$

It is not index 1, since it is singular in $\dot{x}$ and $\dot{y}$.

The approach is now to consider the most differentiated equations and use them and their predecessors to eliminate variables in the other original equations. Equation (4.6b$'$) depends on $\dot{x}$ and $\dot{y}$. Try to solve (4.6b) for x or y. Solving for x, we obtain $x = f_2 - ty$ and $\dot{x} = \dot{f}_2 - y - t\dot{y}$. Substitution into (4.6a) yields $y = \dot{f}_2 - f_1$, in which neither $\dot{x}$ nor $\dot{y}$ appears. Let this equation replace (4.6a) and consider

$$y = \dot{f}_2 - f_1, \tag{4.7a}$$

$$x + ty = f_2 \tag{4.7b}$$

as the new original problem. The modified problem is then simpler than the original problem. In this particular case, it is algebraic, hence index 1, and can be solved immediately. $\square$

The example above demonstrates a more difficult type of index reduction. In contrast to the operations performed by the index reduction algorithm of §3, problems having a rank-deficient Jacobian with respect to the highest-order derivatives will also require that equations be solved symbolically or seminumerically. As the example indicates, this typically entails using an elimination process of Gaussian type. The index reduction algorithm of §3 will reveal whether such a process is necessary. However, we consider developing a full treatment of singular systems beyond the scope of this paper.

5. Numerical results.

We shall now present numerical results for the pendulum problem of Example 3 using the well-known solver DASSL. The numerical solution for various formulations is discussed in [4, pp. 150–157]. There, the evolution of the pendulum equations is computed for a short time interval covering less than a full period. Here we are concerned with the stability and numerical drift of constraints and invariants, thus requiring simulations long enough to reveal secular effects; in all cases, the problem was run for well over a hundred periods. In all the numerical results we use $g = 1$, $m = 1$, and $L = 1$.

In the discussion below, we follow the convention in [4] that the positive direction of the gravitational force is opposite to the direction of the y-axis in the Cartesian coordinate system. Note, however, that for their numerical results to be correct [4, pp. 155–157], we must assume that they have accidentally switched the direction of the y-axis, cf. the obvious misprint in the state-space equation (6.2.5) in [4, p. 151].

The state-space form of the planar pendulum equations is

$$\ddot{\varphi} + \frac{g}{L}\sin\varphi = 0, \tag{5.1}$$

where $x = L\sin\varphi$ and $y = -L\cos\varphi$. This model will be used as a reference model for comparing the numerical results.

Besides studying how well the length constraint is preserved in the numerical solutions, it is also of great interest to study the invariant total energy E, which is the sum of the kinetic energy and the potential energy. The rest state ($\varphi = 0$, $\dot{\varphi} = 0$) is taken as the reference level with $E = 0$. This gives

$$E = \frac{m}{2}L^2\dot{\varphi}^2 + mgL(1 - \cos\varphi).$$

A numerical solution should preserve the energy and keep E constant within numerical accuracy. The expression for the energy can also be used to calculate the amplitude φ_M of the oscillations.

We shall study two cases:

1. Small oscillations with $\varphi(0) = 0.1$ and $\dot{\varphi}(0) = 0.0$, which gives the amplitude $\varphi_M = 0.1$, period time $T \approx 6.29$, and energy $E = 1 - \cos 0.1$.

2. Large oscillations with $\varphi(0) = \pi/2$ and $\dot{\varphi}(0) = -1.0$ as used in [4], which gives an amplitude $\varphi_M = 2\pi/3$, period time $T \approx 8.63$, and energy $E = 1.5$.

The first case is almost linear, whereas the second, with its large amplitude, is strongly nonlinear and will require four dummy pivotings per full period. Whenever DAEs are solved below, the initial conditions are taken to be consistent.

The two problems will be solved for the following formulations:

a. State-space reference model (5.1).

b. Differentiated index-1 model (4.1a″), (4.1b), (4.1c).

c. Stabilized constraint index-2 model [4, pp. 154–155].

d. Dummy derivative index-1 model (4.3), case 1, and (4.2)–(4.3) for case 2.

In all cases the problems were run for 1000 units of time corresponding to approximately 159 periods in case 1, and 116 periods in case 2. Tolerance levels were kept sharp; the parameters ATOL and RTOL in DASSL were taken to be 10^{-9} for all components of the solution except for the model *c*, which cannot be solved with such requirements unless a looser tolerance is used for the two Lagrange multipliers λ and μ. These tolerances were set to 10^{-2}, which effectively corresponds to excluding these algebraic variables from the error tests; cf. [4, p. 156]. Numerical Jacobians were used.

The results for case 1 are displayed in Table 1. Apart from run statistics, we show the deviation ΔE in total energy at the end of the integration interval, and similarly the deviation ΔL in the length constraint. E-drift and L-drift refer to the drifts in E and L, respectively. Note that the length constraint is explicitly present in models *c* and *d*, but it is only an implicit invariant in model *b*. Thus, model *b* shows a drift, and the deviation in the table refers to the error at the end of the integration interval. For models *c* and *d*, there is no drift and the error corresponds to typical deviations throughout the integration. The energy, on the other hand, is an implicit invariant in all four models, resulting in a quite regular drift. For model b, the drift is quadratic in t, whereas the other models exhibit a linear drift. It is to be noted that the drift is less pronounced for models c and d than for the state-space model a.

Results for case 2 are shown in Table 2. Here model d yields the most accurate results, with an energy drift approximately four times smaller than for a, and 25 times smaller than for model c. From the efficiency point of view, the state-space model a is clearly preferable. However, given the rather significant difference in accuracy between models c and d, the modest performance disadvantage of model d seems to be of minor importance.

The results reported here are typical for the pendulum problem. A few other problems from applications in mechanics and electrical engineering have also been tried with good results using the dummy derivative technique.

TABLE 1
Numerical results for small oscillations (case 1) with models a–d.

Model	steps	f-evals	Jac.	ΔE	E-drift	ΔL	L-drift
a	19323	38654	19	$-2.9 \cdot 10^{-7}$	linear	—	—
b	26451	56157	85	$3.3 \cdot 10^{-6}$	quad.	$-3.5 \cdot 10^{-6}$	quad.
c	26697	54774	337	$-1.5 \cdot 10^{-7}$	linear	$\sim 10^{-11}$	none
d	27338	62167	1291	$-1.1 \cdot 10^{-7}$	linear	$\sim 10^{-11}$	none

TABLE 2
Numerical results for large oscillations (case 2) with models a–d.

Model	steps	f-evals	Jac.	ΔE	E-drift	ΔL	L-drift
a	44267	88524	27	$-2.9 \cdot 10^{-6}$	linear	—	—
b	68933	222313	28997	$2.2 \cdot 10^{-3}$	quad.	$1.6 \cdot 10^{-3}$	quad.
c	84087	203850	6545	$1.9 \cdot 10^{-5}$	linear	$\sim 10^{-10}$	none
d	108731	240161	4800	$7.9 \cdot 10^{-7}$	linear	$\sim 10^{-11}$	none

6. Implementation aspects and conclusions. When the proposed index reduction technique is to be implemented, there are two main possibilities. One could either perform all operations completely using symbolic computations in a preprocessing step, or one could employ automatic differentiation [13], [18] to obtain a close integration of the index reduction process and the subsequent numerical treatment. The latter approach seems particularly attractive, since in the augmented system, function evaluations corresponding to differentiated equations must be performed using analytical derivatives that must be continually reevaluated. In addition, the subsequent numerical solution could take advantage of analytical Jacobians. The choice of differentiation technique is probably the most important decision in an implementation of the reduction method. The most important difference between symbolic and automatic differentiation is that the latter can be applied directly to the subprograms defining the DAE. Thus, neither a symbolic representation of the equations nor a code generation step is needed, making automatic differentation much more favorable in our context.

A second important issue is the choice of the matrices $M_i^{[j]}$ by selecting m linearly independent columns from $H_i^{[j]}$. We should aim for well-conditioned matrices, and hence the columns should be selected carefully. An obvious approach is to use a Gram–Schmidt procedure to successively find a set of "maximally" linearly independent columns. However, this approach will depend on the choice of the initial column selected, and it will generally not be possible to find the optimal set.

As far as the practical issues of dummy pivoting are concerned, it is required that one monitors the condition of the matrices $M_i^{[j]}$. Dummy pivoting corresponds to replacing one or more columns of $M_i^{[j]}$ by new columns from $H_i^{[j]}$. It is as yet unclear how this can be carried out inexpensively in large systems. Most likely, it is more demanding to continually monitor the condition than to actually perform the pivoting operation. It should be noted that well-conditioned matrices $M_i^{[j]}$ are needed to ensure that the selected dummy derivatives cancel the exact amount of dynamics in the augmented DAE system, leaving only what corresponds to the dynamics of a state-space form. If the selection is rank deficient, there is a risk that the numerical integration method will get stuck in a singular point. Such effects can readily be seen, e.g., in the pendulum problem; if one inappropriately uses (4.2) for small amplitudes (i.e., when $|x| \ll |y|$), it may very well happen that the integration method cannot get past $x = 0$.

In many cases of practical interest, one will not need a complete automatic reduction procedure, but only the handling of dummy derivatives. Thus, the structure (4.4) is common to all systems described by the Lagrangian equations of the first kind. Therefore, it is sufficient in such applications to select dummy derivatives properly.

The merits of the index reduction technique proposed in this paper lie in the fact that the dummy derivatives are identified and excluded from discretization. As a result, one avoids "over-discretization" of the DAE, and the differentiations inherent in a high-index DAE are carried out analytically rather than numerically. Since the algebraic equations are still present in their original form, there will be no numerical drift away from the solution manifold of the DAE, thus eliminating the need for constraint stabilization. As for invariants that are implicit in both the DAE and the state-space ODE (e.g., energy), the drift is very similar in both formulations. To sum up, the numerical experiments show that, by using the dummy derivative formulation, an accuracy comparable to that of solving a state-space formulation of the problem is obtained. Thus the technique may be considered a viable alternative not only to constraint stabilization, but even to state-space formulations whenever the latter are difficult to obtain.

Acknowledgments. The authors would like to acknowledge helpful discussions on this work with Prof. Steve Campbell and Dr. Bo Bernhardsson.

REFERENCES

[1] T. Alishenas, *Zur numerischen Behandlung, Stabilisierung durch Projektion und Modellierung mechanischer Systeme mit Nebenbedingungen und Invarianten*, Royal Inst. of Tech., Stockholm, Sweden, 1992.

[2] A. Barrlund and B. Kågström, *Analytical and numerical solutions to higher index linear variable coefficient* DAE *systems*, J. Comput. Appl. Math., 31 (1990), pp. 305–330.

[3] J. Baumgarte, *Stabilization of constraints and integrals of motion in dynamical systems*, Comput. Methods Appl. Mech. Engrg., 1 (1972), pp. 1–16.

[4] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, North-Holland, Amsterdam, 1989.

[5] S. Campbell, *The numerical solution of higher index linear time varying singular systems of differential equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 334–348.

[6] ———, *A computational method for general higher index nonlinear singular systems of differential equations*, IMACS Trans. Sci. Comput., 1.2 (1988), pp. 555–560.

[7] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[8] E. Eich, C. Führer, B. Leimkuhler, and S. Reich, *Stabilization and projection methods for multibody dynamics*, Res. Rep. A281, Institute of Mathematics, Helsinki Univ. of Technology, Espoo, Finland, 1990.

[9] C. Führer and B. Leimkuhler, *A new class of generalized inverses for the solution of discretized Euler–Lagrange equations*, in Real-time Integration Methods for Mechanical System Simulation, E. Haug and R. Deyo, eds., Springer-Verlag, Berlin, New York, 1990, pp. 143–154.

[10] ———, *Numerical solution of differential-algebraic equations for constrained mechanical motion*, Numer. Math., 59 (1991), pp. 55–69.

[11] C. Gear, *Differential-algebraic equations, indices, and integral algebraic equations*, SIAM J. Numer. Anal., 27 (1990), pp. 1527–1534.

[12] C. Gear, G. Gupta, and B. Leimkuhler, *Automatic integration of the Euler–Lagrange equations with constraints*, J. Comput. Appl. Math., 12–13 (1985), pp. 77–90.

[13] A. Griewank, D. Juedes, and J. Srinivasan, *Adol-C—a package for the automatic differentiation of algorithms written in C/C++*, Tech. Rep., Argonne National Laboratory, Argonne, IL, 1990.

[14] E. Hairer, C. Lubich, and M. Roche, *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*, Springer-Verlag, Berlin, New York, 1989.

[15] B. Leimkuhler, *Some notes on pertubations of differential-algebraic equations*, Res. Rep. A266, Institute of Mathematics, Helsinki Univ. of Technology, Espoo, Finland, 1989.

[16] S. Mattsson and G. Söderlind, *A new technique for the solution of high index differential-algebraic equations*, in working papers of the 1990 Conf. on the Numerical Solution of Ordinary Differential Equations, 1990.

[17] C. Pantelides, *The consistent initialization of differential-algebraic systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 213–231.

[18] L. Rall, *Automatic Differentiation—Techniques and Applications*, Springer-Verlag, Berlin, New York, 1981.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 693–714, May 1993

ILL-CONDITIONING IN NEURAL NETWORK TRAINING PROBLEMS*

S. SAARINEN[†], R. BRAMLEY[†], AND G. CYBENKO[†‡]

Abstract. The *training* problem for feedforward neural networks is nonlinear parameter estimation that can be solved by a variety of optimization techniques. Much of the literature on neural networks has focused on variants of gradient descent. The training of neural networks using such techniques is known to be a slow process with more sophisticated techniques not always performing significantly better. This paper shows that feedforward neural networks can have ill-conditioned Hessians and that this ill-conditioning can be quite common. The analysis and experimental results in this paper lead to the conclusion that many network training problems are ill conditioned and may not be solved more efficiently by higher-order optimization methods. While the analyses used in this paper are for completely connected layered networks, they extend to networks with sparse connectivity as well. The results suggest that neural networks can have considerable redundancy in parameterizing the function space in a neighborhood of a local minimum, independently of whether or not the solution has a small residual.

Key words. neural network training

AMS(MOS) subject classifications. 65F35, 65K99

1. Introduction.

Some neural network techniques are, in a strictly mathematical sense, an approach to function approximation. As with most approximation methods, they require the estimation of certain (possibly nonunique) parameters which are defined by the problem to be solved [14]. In neural network terminology, finding those parameters is called the *training problem*, and algorithms for finding them are called *training algorithms*. This nomenclature comes from analogy with biological systems, since a set of inputs to the function to be approximated are presented to the network, and the parameters are adjusted to make the output of the network close in some sense to the known value of the function.

Feedforward neural networks use a specific parameterized functional form to approximate a desired input/output relation. Typically, a system is sampled resulting in a finite set of pairs $(t, \tau) \in \mathbb{R}^p \times \mathbb{R}$ where the first coordinate is a position in p-dimensional space and the second coordinate refers to the assigned value for the point. The feedforward neural network function, also from $\mathbb{R}^p \mapsto \mathbb{R}$, has a set of parameters, called weights, which must be determined so that the input and output values as given by the sample data are matched as closely as possible by the approximating neural network. The neural network function for the ith input pattern $(i = 1, 2, \ldots, m)$ can be written succinctly in the form

$$F = F(t_i, \mathbf{x}), \tag{1}$$

where t_i is a p-dimensional input vector, and where $\mathbf{x} \in \mathbb{R}^n$ is the weight vector of parameters to be determined. The form of the function $F(t_i, \mathbf{x})$ for a feedforward network is presented and derived in §3. Let the desired value (or output) for the ith input be denoted by τ_i. A common formulation of the training problem is to find

* Received by the editors February 11, 1991; accepted for publication (in revised form) May 18, 1992.

† Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

‡ This author's work was supported in part by National Science Foundation grant MIP-89-11025, Air Force Office of Scientific Research/Defense Advanced Research Projects Agency contract 89-0536, CCR 900000N, and Department of Energy grant DE-FG02-85ER25001.

values of $\mathbf{x}$ such that the norm of $f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))^T$ is minimized, where f_i denotes the residual between the value of the approximating function and the desired value

$$f_i(\mathbf{x}) = F(t_i, \mathbf{x}) - \tau_i.$$

Often the l_2-norm is used in the minimization of such a function, and we obtain a least squares problem

$$\|f(\mathbf{x}^*)\|^2 = \min_{\mathbf{x}\in\mathbb{R}^n} \sum_i f_i(\mathbf{x})^2. \tag{2}$$

A numerical minimization algorithm is applied to this equation in order to find a suitable $\mathbf{x}^*$. This minimization is called training.

The current interest in using feedforward neural networks for pattern recognition problems has revealed that the training algorithms are computationally time consuming for a large class of algorithms [17]. Examples of these algorithms are backpropagation (which can be classified as a steepest descent algorithm), conjugate gradient algorithms, and other nonlinear optimization algorithms for parameter estimation. However, little research has been directed into the question of understanding why the training algorithms are slow to converge on neural networks even though the underlying techniques often perform very well for other problems. This paper addresses the question of why these algorithms converge slowly by showing that rank-deficiencies may appear in the Jacobian for a neural network, making the problem numerically ill conditioned.

Except for pattern search methods, the convergence properties of optimization algorithms for differentiable functions depend on properties of the first and/or second derivatives of the objective function [9]. For example, steepest descent explicitly requires the first derivative to define its search direction, and implicitly relies on the second derivative whose properties govern the rate of convergence. When optimization algorithms converge slowly (or not at all) for neural network problems, this suggests that the underlying derivative matrices are numerically ill conditioned. The obvious questions to ask in this case are:

- Will using a higher-order method help avoid this problem? For example, will a quasi-Newton or Newton method reduce the number of iterations?
- Is the mathematical formulation of the training problem the "correct" one? For example, by changing the parameterization or scaling of the problem, can the training algorithm be made more rapidly convergent?
- Is the difficulty of solving the training problem an intrinsic feature of the neural network itself, and not an artifact of the problem formulation and chosen training algorithm?

The proliferation of neural network techniques makes it impossible to answer these questions for all types of networks and all types of problems the networks are to solve. This paper concentrates on one type of network, a multilayer feedforward network, and one type of problem, approximating indicator functions of sets in the plane. These choices were made because multilayer feedforward networks are commonly used by researchers and the classification problem is one for which neural nets are potentially suitable; see [3], [15], and [8]. Furthermore, the training problems examined here are primarily *overdetermined*, that is, the number of training data points is greater than or equal to the number of network parameters. This seems to match the intended usage of neural networks, since for problems such as speech recognition or classification problems the amount of training data available exceeds the number of parameters that can be accommodated in a practical neural network. The results, however, also seem to apply to underdetermined problems.

It is important to observe that the rank-deficiency shown and explained in the subsequent analysis is independent of the size of the residual. Therefore, the fact that a network configuration cannot exactly "solve" a classification problem does not mean that the network parameterization is parsimonious or not redundant. In fact, rank-deficiency is an indication that redundancy does occur locally. As a simple example, consider using m polynomials of degree n or less, $n < m$, to perform a least squares fit on $m+1$ or more data points in general position: the residual will generically be nonzero, but the function class is inherently overparameterized. This redundancy is usually considered an advantage of neural networks, allowing some degree of fault tolerance.

In this paper we will first review nonlinear least squares problems; second, we will describe further the neural network problem and show general properties of the Jacobian for a feedforward neural network; and finally, we will investigate the Jacobian and Hessian for some examples. The analytic results about the Jacobian show that it can be rank-deficient in certain situations that can be enumerated. We then show that this rank-deficiency actually occurs in a number of experiments. For methods that use Hessian information explicitly, the rank-deficiency or ill-conditioning of the Jacobian still plays a role because the Jacobian part of the Hessian is dominant. We emphasize that this paper deals only with the least squares formulation of a training algorithm, and that all the results obtained are for this class of problems.

2. Review of nonlinear least squares. The material in this section establishes notation and describes the difficulties imposed by a rank-deficient or ill-conditioned Jacobian. We define the rank and the condition number of a matrix $A \in \mathbb{R}^{l\times q}$ $(l \geq q)$ by the singular value decomposition $A = U\Sigma V^T$, where $U^TU = I_l$, $V^TV = I_q$, and $\Sigma \in \mathbb{R}^{l\times q}$ is a diagonal matrix $\text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_q)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_q \geq 0$. A has rank $r < q$ if $\sigma_{r+1} = \cdots = \sigma_q = 0$ and $\sigma_r \neq 0$. The *degree of rank-deficiency* of A is $q - r$. The *condition number* of A is $\kappa = \kappa(A) = \sigma_1/\sigma_r$, and A is *ill conditioned* if $\kappa(A)$ is "large." Except in special cases, in practice it is rare to find singular values exactly equal to zero, and numerically determining r is difficult (see, for example, Figs. 6–8). Because of this the numerical examples of this paper will take $q = r$, but the analytic descriptions will also discuss the $q < r$ case. Note that if $\sigma_{r+1}, \ldots, \sigma_q > 0$ are all very small relative to σ_1, A is close to being rank-deficient of degree at least $q - r$.

Letting $\phi(\mathbf{x}) = \frac{1}{2} \parallel f(\mathbf{x}) \parallel^2$ in (2), the gradient of ϕ is

$$\nabla\phi(\mathbf{x}) = J(\mathbf{x})^T f(\mathbf{x}), \tag{3}$$

and the Hessian matrix of ϕ is

$$H(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) H_i(\mathbf{x}), \tag{4}$$

where $[J_{ij}] = \partial f_i/\partial x_j$ is the Jacobian matrix of $f(\mathbf{x})$ and $H_i(\mathbf{x})$ is the Hessian matrix of the component function $f_i(\mathbf{x})$. For convenience, the explicit dependence on $\mathbf{x}$ will sometimes be omitted by writing $H = J^TJ + \sum_{i=1}^m f_iH_i$, etc. Algorithms for minimizing $\phi(\mathbf{x})$ usually take advantage of the special structure of $\nabla\phi(\mathbf{x})$ and $H(\mathbf{x})$; the reader is referred to [5] for a survey of such algorithms.

A given point $\mathbf{x}^*$ is a *critical point* of ϕ if $\nabla\phi(\mathbf{x}^*) = 0$, and $\mathbf{x}^*$ is a local minimum only if all the eigenvalues of $H(\mathbf{x}^*)$ are nonnegative. One of the weaknesses of commonly used neural network training algorithms is the failure to check the obtained "solution" for optimality, except in the trivial consistent case where $f(\mathbf{x}^*) = 0$. Optimization methods for solving the nonlinear least squares problem generate a search direction $\mathbf{p}$ and a stepsize α from the current iterate $\mathbf{x}$. The search direction and stepsize are usually chosen so that $\phi(\mathbf{x}+\alpha\mathbf{p}) < \phi(\mathbf{x})$. Table 1 shows the search directions used by some common optimization methods, assuming that J is full rank.

TABLE 1
Search directions used by various optimization methods.

Algorithm	Search direction
Steepest descent	$-J^T f$
Conjugate gradient	$-J^T f + \beta\tilde{p}$, with $\tilde{p}$ = previous search direction β = a scalar
Newton	$-(J^T J + \sum_{i=1}^m f_i H_i)^{-1} J^T f$
Gauss–Newton	$-(J^T J)^{-1} J^T f$
Levenberg–Marquardt	$-(J^T J + \rho_k I)^{-1} J^T f$
Quasi-Newton	$-(J^T J + B_k)^{-1} J^T f$, B_k satisfying a quasi-Newton condition

2.1. Local convergence. The local convergence properties of the methods in Table 1 depend on the size of the residual $f(\mathbf{x}^*)$ and the rank and condition number of $H(\mathbf{x}^*)$, the Hessian at the solution. Steepest descent has a q-linear rate of convergence (see [11] for a definition of q-linear) with an asymptotic error constant proportional to $(\kappa - 1)/(\kappa + 1)$, where κ is the condition number of $H(\mathbf{x}^*)$ [9]. Conjugate gradient methods generally have a linear rate of convergence, but their behavior depends on the definition of the conjugacy scalar β as well as the frequency of *restart*, i.e., reinitialization of the algorithm [13]. Quasi-Newton methods have a superlinear rate of convergence if, roughly speaking, $H(\mathbf{x}^*)$ is nonsingular and the matrices B_k are chosen so that $J^T J + B_k$ approximates $H(\mathbf{x}^*)$ along the search directions; see [4] for a more detailed description of both quasi-Newton methods and their convergence properties. Newton's method has a quadratic rate of convergence, provided that the Hessian is nonsingular at $\mathbf{x}^*$. The other methods have local convergence properties that can be seen by considering them as approximations to Newton's method. For example, if the residual is zero at $\mathbf{x}^*$, then $H(\mathbf{x}^*) = J^T(\mathbf{x}^*)J(\mathbf{x}^*)$ and the Gauss–Newton method shares the quadratic convergence rate of Newton's method if $J(\mathbf{x}^*)$ is full rank. Under the same conditions the Levenberg–Marquardt method has quadratic convergence when the ρ_k in Table 1 is zero; in practice, ρ_k is chosen to provide better global convergence. When the residual is large, however, Gauss–Newton and Levenberg–Marquardt methods have a linear convergence rate.

2.2. Global convergence. Minimization algorithms can spend the majority of their time in finding a neighborhood of the solution in which local convergence theorems apply, so the local convergence rate is irrelevant if the algorithm fails to enter that neighborhood or if it is extremely small. Note the critical dependence of all of the methods in Table 1 on the Jacobian J; each simply multiplies the *fundamental* search direction $-J^T f$ by some matrix estimating second-order information (for steepest descent the matrix is I) except for the conjugate direction, which has a linear combination of the current and previous vectors $-J^T f$ as its search direction. The search directions for the Newton or Gauss–Newton methods are ill defined when the Hessian is singular or the Jacobian is not full rank, respectively, but the Levenberg–Marquardt parameter ρ_k and the quasi-Newton matrix B_k are normally chosen to assure that their respective search directions are well defined. Unfortunately, if J or the Hessian are rank-deficient or ill conditioned at many of the iterate points, the methods can actually perform worse than steepest descent, which in turn can fail to converge in finite precision arithmetic.

2.3. Regularization. When J is rank-deficient or ill conditioned, these algorithms can be generalized by two common regularization approaches. The first replaces J with $J_r = U\Sigma_r V^T$, where $\Sigma_r = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0, \ldots, 0)$, $J = U\Sigma V^T$ is the singular value decomposition of J, and r is an estimate of the rank of J as discussed above. However, the computational determination of r is a difficult problem, and can have a dramatic effect on the resulting search direction (see the example in [6, p. 136]). Furthermore, the resulting search direction has nonzero components only in a subspace of dimension r. When $r \ll n$ and the subspace changes slowly from iteration to iteration, the method can fail to make sufficient progress to a minimum. Methods for circumventing this difficulty generally add a component in the orthogonal complement of the subspace of dimension r to the search direction, as in [6]. A second regularization approach adds a small multiple $\gamma \parallel \mathbf{x} \parallel$ of the norm of the weight vector to the objective function, possibly allowing γ to change on each iteration. This restricts the size of the weights, but unfortunately the weights frequently need to be large for accurate learning, i.e., a good least squares solution. So if γ is prevented from decreasing to zero the quality of solution can suffer, while if γ is decreased to zero eventually the same ill-conditioning problems are encountered. Nevertheless, the Levenberg–Marquardt algorithm used in §3.4 implicitly uses this second form of regularization (see [10]), and can provide adequate solutions in many cases.

For most overdetermined nonlinear least squares problems, these considerations are minor. Generally the Jacobian is full rank (but ill-conditioning can occur) and it is only at exceptional points that J is rank-deficient, but even then the rank-deficiency is small. For neural network problems, however, we show that a large number of columns of the Jacobian can easily be nearly linearly dependent and so the matrix is close in 2-norm to a matrix with a large degree of rank-deficiency.

3. Neural network training problems. A neural network consists of three types of computational elements or nodes arranged in layers: input layer nodes, hidden layer nodes, and output layer nodes with weighted connections between them. Each node has *several* inputs and only *one* output. A feedforward neural network has connections between neighboring layers with information flowing only in one direction. There are no connections within a layer. We further use networks which are fully connected between layers, that is, every node in a given layer has a directed connection with all the nodes in the previous layer and in the successive layer. Such a neural network with two hidden layers is depicted in Fig. 1. The flow of information is upwards in the figure. The input layer nodes and the output layer nodes are depicted by triangles, and the hidden layer nodes by circles. All arrows leaving a given node denote the unique output for the node. A hidden layer node forms its output o by first forming the weighted sum of its inputs u_i,

$$s = \sum_{i=1}^{j} x_i u_i + x_0,$$

where the $x_i, i = 0, \ldots, j$ are called weights and x_0 is called an offset. The hidden node then applies a univariate excitation function $\sigma(s)$ to the weighted sum s and this value, $o = \sigma(s)$, becomes the output of the node. The input layer and output layer nodes do not apply an excitation function to their inputs. Therefore, the ***inputs*** in Fig. 1 are the same as the network inputs (shown as v_1 and v_2). A more elaborate description of neural networks in general can be found in [15]. We will next derive the functional form of the network function.

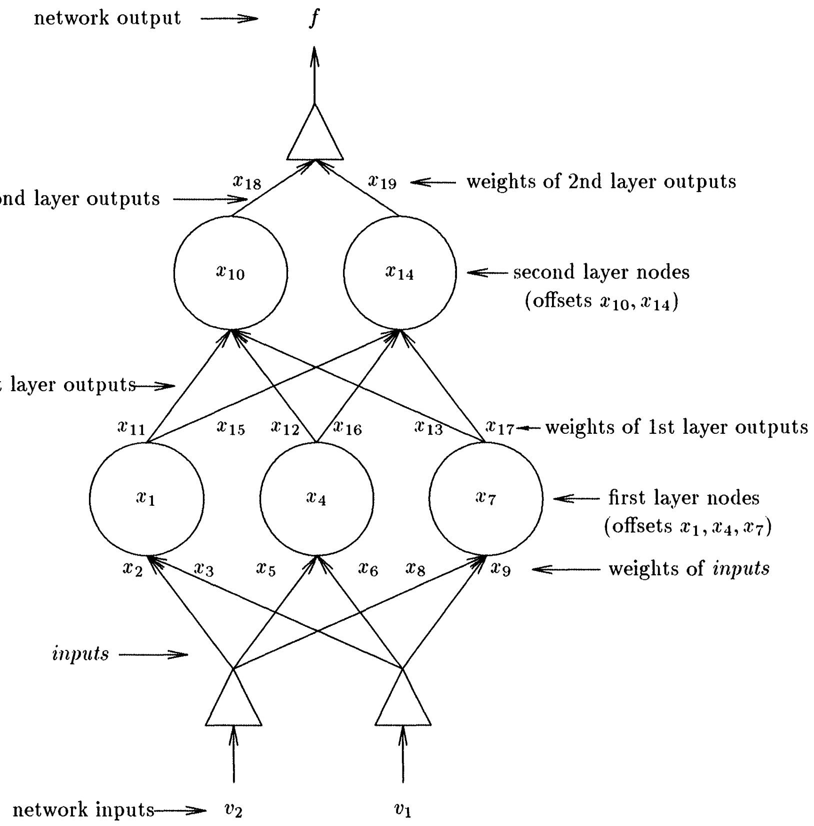

network output
f
Q_i second layer outputs
x_{18}
x_{19}
weights of 2nd layer outputs
x_{10}
x_{14}
second layer nodes (offsets x_{10}, x_{14})
P_j first layer outputs
x_{11}
x_{15}
x_{12}
x_{16}
x_{13}
x_{17}
weights of 1st layer outputs
x_1
x_4
x_7
first layer nodes (offsets x_1, x_4, x_7)
x_2
x_3
x_5
x_6
x_8
x_9
weights of *inputs*
inputs
network inputs
v_2
v_1

FIG. 1. *A 2-3-2 feedforward neural network with the weights indicated by $x_i, i = 1, \ldots, 19$. The offsets are written inside the nodes (indicated by circles). The quantities P_j and Q_i are discussed in the text.*

Let the number of inputs to the network be h, the number of first layer nodes be p, and the number of second layer nodes be s. The network can then be concisely labeled as $h - p - s$ and Fig. 1 represents a 2-3-2 network. Let the total number of training data points be m and the total number of parameters (i.e., weights associated with the arcs and offsets in the nodes) be $n = p(h + s + 1) + 2s$.

For the ith training data input pattern we write

$$t_i = (v_1^{(i)}, v_2^{(i)}, \ldots, v_h^{(i)}),$$

and let weights associated with the input layer arcs be $x_{(j-1)(h+1)+1+k}$, $1 \le j \le p$ and $1 \le k \le h$, and the offsets associated with the first layer nodes be $x_{(j-1)(h+1)+1}$, where $1 \le j \le p$.

Then we can write for the output of the jth first layer node for the training data point i

$$\sigma(P_j^{(i)}) = \sigma\left(\sum_k x_{(j-1)(h+1)+1+k}\, v_k^{(i)} + x_{(j-1)(h+1)+1}\right), \tag{5}$$

where σ is a sigmoidal or excitation function. Similarly, for the second layer nodes let the weights on the arcs be $x_{t+(l-1)(f+1)+1+j}$, where $t = p(h+1)$, $1 \le l \le s$ and $1 \le j \le p$, and the offsets be $x_{t+(l-1)(p+1)+1}$, where $1 \le l \le s$. The output of the lth second layer node for training data point i is

$$\sigma(Q_l^{(i)}) = \sigma\left(\sum_j x_{t+(l-1)(p+1)+1+j}\, \sigma(P_j^{(i)}) + x_{t+(l-1)(p+1)+1}\right). \tag{6}$$

Furthermore, if the weights on the last layer are x_{u+l}, where $1 \le l \le s$ and $u = t + s(p+1)$, then the output of the network for training data point i is

$$F(t_i, \mathbf{x}) = \sum_l x_{u+l}\, \sigma(Q_l^{(i)}). \tag{7}$$

In this paper we assume that the node on the last layer will not apply an excitation function to its inputs (this will not make the problem less general but rather avoid using one extra parameter). The aim is to find the weights on the arcs and the offsets by minimizing (2) and using some input-output pairs of the patterns that the network should approximate.

The form of the excitation function can vary. Examples include

$$\sigma_1(x) = \frac{1}{1+e^{-x}} \quad \text{and} \quad \sigma_2(x) = \frac{2}{\pi}\tan^{-1} x. \tag{8}$$

The functions $\sigma_1(x)$ and $\sigma_2(x)$ are called sigmoidal functions, and their ranges are $(0,1)$ and $(-1,1)$, respectively. Note that both their derivatives approach 0 when $|x| \gg 1$. The general form of these functions can be obtained by making the transformation $x \mapsto \alpha x + \beta$, but the ranges of the functions and their general form stay the same. It has been shown that under very general conditions on the activation functions, such classes of neural networks as in (7) are universal approximators [3]. The main reason for choosing the two functions in (8) is that they can approximate the hard delimiter step function and are continuously differentiable. In this paper we will use $\sigma_1(x)$, but the method in this paper can be used to derive similar results for other excitation functions as well (including radial basis functions [2], [12]).

3.1. Explicit form of the Jacobian for a two-layer network. The Jacobian for the multilayer feedforward network problem can be written explicitly using the notation from the previous section. The Jacobian is a matrix of size $m \times n$, where each row of the Jacobian corresponds to a single training data point. We present the Jacobian layer by layer and specifically show the components for the arc weights separately from the offsets. Thus we will present a row of the Jacobian, J, but will show the row in blocks (the first subscript refers to the training data point i, and the

TABLE 2
Summary of the Jacobian.

Jacobian element	Formula	Corresponding unknowns
$J_{i,(j-1)(h+1)+1}$	$\sigma'(P_j^{(i)})\sum_l x_{u+l}x_{t+(l-1)(p+1)+1+j}\ \sigma'(Q_l^{(i)})$	First layer offsets
$J_{i,(j-1)(h+1)+1+k}$	$v_k^{(i)}\sigma'(P_j^{(i)})\sum_l x_{u+l}x_{t+(l-1)(p+1)+1+j}\ \sigma'(Q_l^{(i)})$	First layer weights
$J_{i,t+(l-1)(p+1)+1}$	$x_{u+l}\sigma'(Q_l^{(i)})$	Second layer offset
$J_{i,t+(l-1)(p+1)+1+j}$	$x_{u+l}\sigma(P_j^{(i)})\sigma'(Q_l^{(i)})$	Second layer weights
$J_{i,u+l}$	$\sigma(Q_l^{(i)})$	Last layer weights

second to the actual parameter or column of the Jacobian). The form of the resulting Jacobian is summarized in Table 2 and the third column in the table explains the origin of the term.

We can now write the ith row of the Jacobian (using its inherent block structure) for the 2-3-2 network in Fig. 1:

$$\begin{aligned}
J_{i,1\ldots3} &= [x_{18}x_{11}\sigma'(Q_1^{(i)}) + x_{19}x_{15}\sigma'(Q_2^{(i)})]\sigma'(P_1^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T,\\
J_{i,4\ldots6} &= [x_{18}x_{12}\sigma'(Q_1^{(i)}) + x_{19}x_{16}\sigma'(Q_2^{(i)})]\sigma'(P_2^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T,\\
J_{i,7\ldots9} &= [x_{18}x_{13}\sigma'(Q_1^{(i)}) + x_{19}x_{17}\sigma'(Q_2^{(i)})]\sigma'(P_3^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T,\\
J_{i,10\ldots13} &= x_{18}\sigma'(Q_1^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T,\\
J_{i,14\ldots17} &= x_{19}\sigma'(Q_2^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T,\\
J_{i,18\ldots19} &= (\sigma(Q_1^{(i)}), \sigma(Q_2^{(i)}))^T.
\end{aligned}$$

Here $J_{i,j}$ denotes the element in the ith row and jth column of the Jacobian. From the dependency on i in each column of J, one row cannot be an exact linear combination of sets of rows. If there are some linear or near-linear dependencies between rows or columns in the Jacobian they must arise in some other manner.

3.2. Properties of the sigmoidal. An explanation of the rank-deficiency of the Jacobian relies on the properties of sigmoidal functions and their derivatives. We consider the sigmoidal function $\sigma(x) = \sigma_1(x)$ in (8), with derivative

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}.$$

The columns of the Jacobian for the neural network function has terms of the form $\sigma(x)$, $\sigma'(x)$, $\sigma'(x)\sigma(y)$, and $\sigma'(x)\sigma'(y)$ with constant coefficients. To explain the near linear dependence between columns of J, we are therefore interested in the quantities

$$\begin{aligned}
\mathrm{A}(x,y) &= \sigma(x+y) - \sigma(x), & \mathrm{B}(x,y) &= \sigma'(x+y) - \sigma'(x),\\
\mathrm{C}(x,y) &= \sigma'(x+y)\sigma'(x), & \mathrm{D}(x,y) &= \sigma(x+y)\sigma'(x),
\end{aligned}$$

where $x \in [-\rho, \rho]$ and $y \in [-\delta, \delta]$.

Figure 2 contains the graph for $\mathrm{A}(x,y)$ for various values of $x \in [-10, 10]$ and as a function of $y \in [-20, 20]$. Note that the differences in the graphs for large values of

$|x|$ are small. Figure 3 shows the graph for $\mathrm{B}(x,y)$ for $x \in [-10,10]$ and $y \in [-20,20]$. Note that $\mathrm{B}(x,y)$ varies slowly with x. It is easy to see that $\sup_{x,y} \mathrm{A}(x,y) = 1$ and $\sup_{x,y} \mathrm{B}(x,y) = 0.25$ from the properties of σ and σ'. Similarly, $|\mathrm{C}(x,y)| \leq \frac{1}{16}$ and $|\mathrm{D}(x,y)| \leq \frac{1}{4}$, for any x and y. We show the form of these functions in Figs. 4 and 5. The upper limits of these functions are not as important as the fact that often the values of these functions are close to 0 for a large range of x and y.

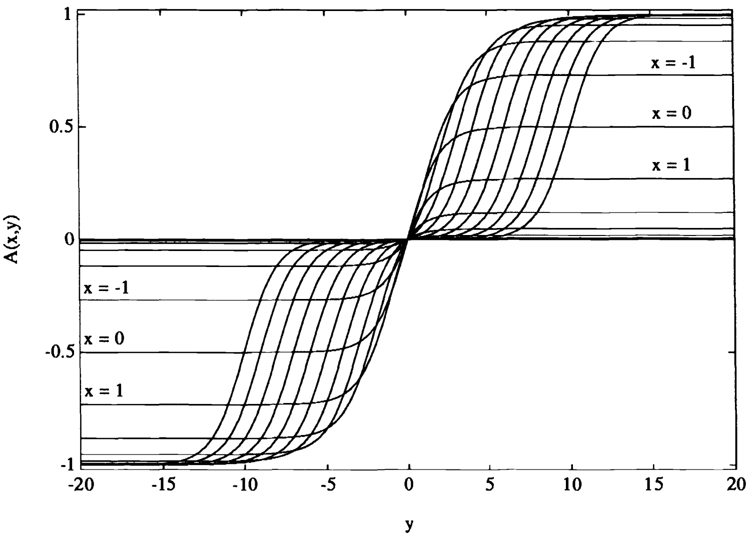


FIG. 2. *Function* $\mathrm{A}(x,y)$ *for various* x. *In the figure we have plotted curves for* $x = -10, -9, \ldots, 0, \ldots, 9, 10$, *and three are indicated.*

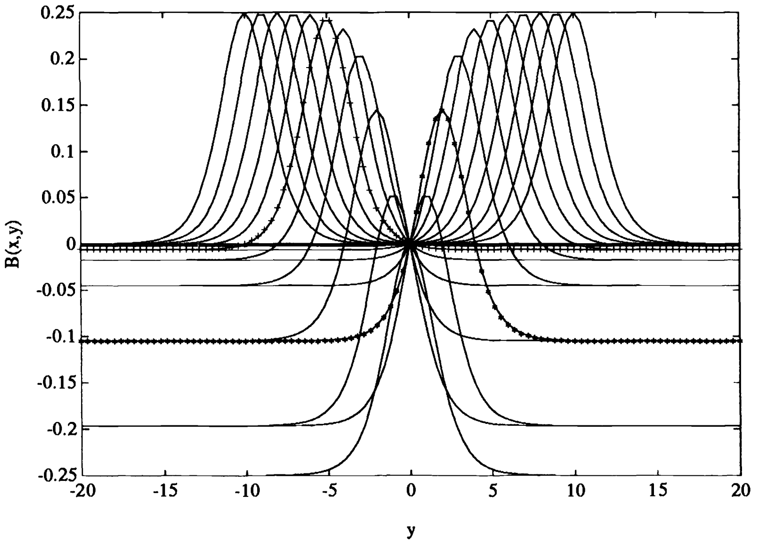


FIG. 3. *Function* $\mathrm{B}(x,y)$ *for various* x. *We have used* $x = -10, -9, \ldots, 9, 10$ *and indicated the curves for* $x = 5$ *with* $+$ *(on the left-hand side of the figure) and* $x = -2$ *with* $*$ *(on the right-hand side).*

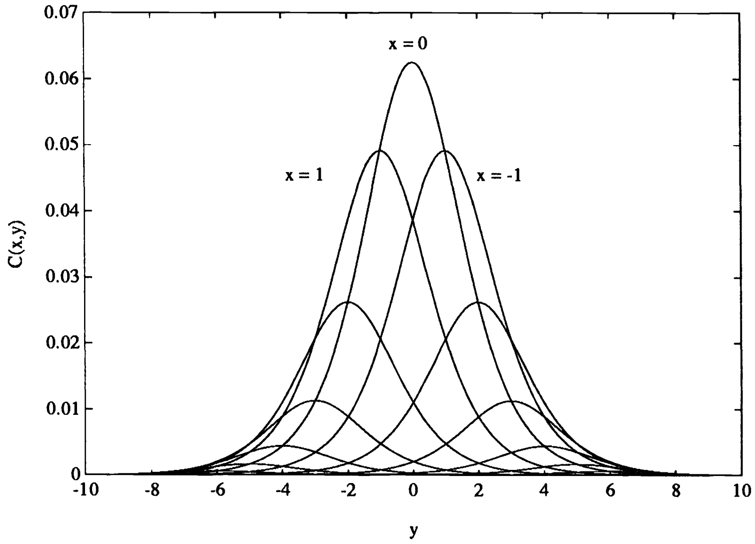


FIG. 4. *Function* $C(x, y)$ *for* $x = -10, -9, \ldots, 9, 10$. *The curves for* $x = -1, 0,$ *and* 1 *are indicated.*

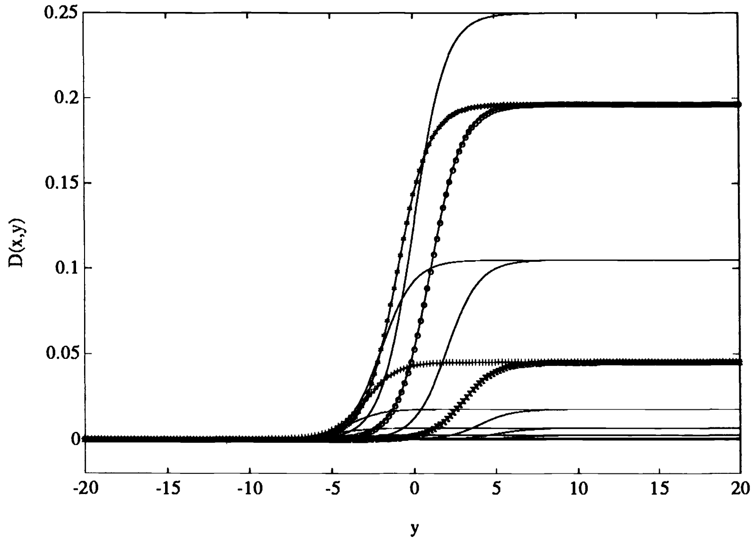


FIG. 5. *Function* $D(x, y)$ *for* $x = -10, -9, \ldots, 9, 10$. *The following curves are indicated:* $x = -1$ (*o*), $x = 1$ (***), $x = -3$ (x), *and* $x = 3$ (+).

3.3. Rank-deficiency of the Jacobian of a two hidden layer network. This section analyzes five situations where the Jacobian for a two hidden layer feed-forward network is ill conditioned or rank-deficient. These results easily extend to a network with more hidden layers and follow from three simple propositions.

PROPOSITION 3.1. *Let* B *be a submatrix of* A *consisting of columns of* A. *Then* $\kappa(B) \leq \kappa(A)$.

Proof. This follows immediately from a repeated application of [7, Corollary 8.3.3]. □

PROPOSITION 3.2. *Let $A = [x, y] \in \mathbb{R}^{n\times 2}$, and suppose that the angle θ between x and y satisfies $\cos(\theta) = 1 - \epsilon$ for some $\epsilon \in (0,1)$. Then*

$$\kappa^2(A) \geq \frac{1}{4\epsilon(2-\epsilon)} \cdot \left(\frac{\| y \|^2}{\| x \|^2} + \frac{\| x \|^2}{\| y \|^2} + 2 \right).$$

Proof. As noted before, the singular values of A are the square roots of the eigenvalues of $G = AA^T$. The eigenvalues of G are nonzero and satisfy

$$\lambda = \frac{1}{2}[x^Tx + y^Ty \pm \sqrt{(x^Tx - y^Ty)^2 + 4(x^Ty)^2}].$$

Using this and completing the square in $\kappa(G) = \lambda_{\max}/\lambda_{\min}$ gives

$$\kappa(G) = \frac{[x^Tx + y^Ty + \sqrt{(x^Tx - y^Ty)^2 + 4(x^Ty)^2}]^2}{4(x^Tx \cdot y^Ty - (x^Ty)^2)}.$$

Since $\cos(\theta) = \frac{x^Ty}{\|x\|\cdot\|y\|} = 1 - \epsilon$, $(x^Ty)^2 = (1-\epsilon)^2 x^Tx \cdot y^Ty$ and so

$$\kappa(G) \geq \frac{(x^Tx + y^Ty)^2}{4[1 - (1-\epsilon)^2]x^Tx \cdot y^Ty} = \frac{1}{4\epsilon(2-\epsilon)} \cdot \left(\frac{\| y \|^2}{\| x \|^2} + \frac{\| x \|^2}{\| y \|^2} + 2 \right). \qquad \square$$

Proposition 3.2 provides a lower bound on the condition number of a matrix when two columns are nearly collinear. In fact, if some number of columns of a matrix, say p, are almost collinear, then one should expect at least $p - 1$ singular values of the matrix to be relatively small. Thus the matrix would almost have a rank-deficiency of $p - 1$ in such a case. These claims can be made precise along the lines of Proposition 3.2, but now we assume that the columns are normalized.

PROPOSITION 3.3. *Let $A = [a_1, \ldots, a_p]$ be an $m \times p$ matrix with normalized columns, so that $a_i^T a_j = 1 - \epsilon_{ij}$ for all $1 \leq i, j \leq n$, with $\epsilon_{ii} = 0$ for $i = 1, 2, \ldots, p$. Suppose that $\epsilon_{ij} \leq \epsilon < 1$. Then A has one singular value at least as large as $\sqrt{p(1-\epsilon)}$ while the remaining singular values are no larger than $\sqrt{p\epsilon}$.*

Proof. By assumption, $A^TA = B - E$ where B is the matrix of all ones and $E = [\epsilon_{ij}]$. The eigenvalues of B are p (with multiplicity 1) and 0 (with multiplicity $p-1$), while the eigenvalues of E are no larger than $p\epsilon$ by the Gersgorin disk theorem. An application of the Wielandt–Hoffman theorem [7] shows that A^TA has one eigenvalue λ_p satisfying $|\lambda_p - p| \leq p\epsilon$, and the other eigenvalues λ_i satisfy $|\lambda_i| \leq p\epsilon$. Since the nonzero singular values of A are the square roots of the nonzero eigenvalues of A^TA, $\sigma_1 \geq \sqrt{p(1-\epsilon)}$ and $\sigma_i \leq \sqrt{p\epsilon}$ for $i = 2, 3, \ldots, p$. $\square$

The preceding results quantify the following argument. For a Jacobian J to be ill conditioned, it suffices to have two columns that are nearly dependent. If the cosine of the angle between those two columns is $1 - \epsilon$, then the condition number of J is at least $\mathcal{O}(\epsilon^{-1/2})$. If p of the columns of J are nearly multiples of each other (that is, the cosine of the angle between any pair of the p vectors satisfies $|\cos\theta| \geq 1 - \epsilon$) and the columns are normalized, then at least $p - 1$ singular values of J are $\mathcal{O}((p\epsilon)^{1/2})$, and one is $\mathcal{O}(p^{1/2})$. This implies that J is within $p\epsilon$ in the 2-norm to a matrix with degree of rank-deficiency at least $p-1$, which causes difficulties for linear least squares solvers.

To simplify the notation, define

$$\underline{\sigma}(Q_r) = (\sigma(Q_r^{(1)}), \sigma(Q_r^{(2)}), \ldots, \sigma(Q_r^{(m)}))^T$$

and define the vectors $\underline{\sigma}'(Q_r)$, $\underline{\sigma}(P_r)$, and $\underline{\sigma}'(P_r)$ similarly. We now describe five cases where conditions internal to the network lead to ill-conditioned and close to rank-deficient matrices.

Case 1. If for some j, $\underline{\sigma}(P_j)$ is a multiple of $e = (1, 1, \ldots, 1)^T$, then any pair of columns in the Jacobian corresponding to an arc and an offset, which have $\sigma(P_j^{(i)})$ as input, on the second layer will produce two identical columns of J. If this holds for K first layer nodes, then $K+1$ columns of J are identical. Since there are s second layer nodes, this gives a rank-deficiency of sK. Note that this condition is approximately satisfied if the angle between e and $\underline{\sigma}(P_j)$ is small.

Case 2. If $\underline{\sigma}'(Q_k)$ and $\underline{\sigma}'(Q_j)$ are multiples of each other then the block of columns in J corresponding to the parameters of the nodes k and j of the second layer are identical, producing a rank-deficiency of $(p+1)+(h+1)$. Note that this condition is approximately satisfied if the angle between $\underline{\sigma}'(Q_k)$ and $\underline{\sigma}'(Q_j)$ is small. The term $p+1$ appears from the second layer nodes (and parameters associated with them), and the term $h+1$ appears from the Jacobian calculated with respect to first layer parameters. This is shown in (9) for $k=1$ and $j=2$:

$$(9)\qquad \begin{aligned} J_{i,10\ldots13} &= x_{18}\ \sigma'(Q_1^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T, \\ J_{i,14\ldots17} &= x_{19}\ \sigma'(Q_2^{(i)})(1, \sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T. \end{aligned}$$

A rank-deficiency of $p+1=4$ occurs because the block array $[e, \underline{\sigma}(P_1), \underline{\sigma}(P_2), \underline{\sigma}(P_3)]^T$ is always identical in $J_{i,10\ldots13}$ and $J_{i,14\ldots17}$. In general, if $\text{rank}[\underline{\sigma}'(Q_1), \underline{\sigma}'(Q_2), \ldots, \underline{\sigma}'(Q_s)] = L$, then the rank-deficiency is $(s-L)(p+1+h+1)$.

Case 3. If $\underline{\sigma}'(P_k)$ and $\underline{\sigma}'(P_j)$ are multiples of each other then the block of columns corresponding to the first layer node k of J (i.e., the arcs and offset parameters) is a multiple of the block of columns corresponding to the first layer node j. Note that this condition approximately holds if $\underline{\sigma}'(P_k)$ and $\underline{\sigma}'(P_j)$ have a small angle between them. As in Case 2 above, the deficiency is $h+1$ because the block array $[e, \underline{v}_1, \underline{v}_2]^T$ is repeated in $J_{i,1\ldots3}$ and $J_{i,4\ldots6}$. More generally, if $\text{rank}[\underline{\sigma}'(P_1), \underline{\sigma}'(P_2), \ldots, \underline{\sigma}'(P_f)] = L$ then the rank-deficiency from this condition is $(p-L)(h+1)$.

Case 4. If $\underline{\sigma}(Q_k)$ and $\underline{\sigma}(Q_j)$ are multiples of each other then the columns corresponding to last layer parameters k and j are linearly dependent. This is shown in (10) for $k=1$ and $j=2$:

$$(10)\qquad J_{i,18\ldots19} = (\sigma(Q_1^{(i)}),\ \sigma(Q_2^{(i)}))^T.$$

More generally, if $\text{rank}[\underline{\sigma}(Q_1), \underline{\sigma}(Q_2), \ldots, \underline{\sigma}(Q_s)] = L$ then the rank-deficiency is $s-L$.

There is also a possible type of dependency in the columns of the Jacobian similar to Case 4 above, but which in fact rarely occurs.

Case 5. If $\underline{\sigma}(P_k)$ and $\underline{\sigma}(P_j)$ are multiples of each other, but $\underline{\sigma}(P_k)$ is not a multiple of e (so that Case 1 is excluded), then the columns corresponding to the arcs k and j are multiples of each other. More generally, if $\text{rank}[\underline{\sigma}(P_1), \underline{\sigma}(P_2), \ldots, \underline{\sigma}(P_f)] = L$ then the rank-deficiency from this condition is $(p-L)s$.

Finally, note that rank-deficiency can arise in less restrictive ways than indicated by the hypotheses above. For example, in Case 3 it is only necessary that $\sigma'(P_k)$ and $\sigma'(P_j)$ nearly be multiples of each other for those components i with $\sigma'(Q_1^{(i)})$ and $\sigma'(Q_2^{(i)})$ to be nonzero. Because the derivative of the sigmoidal is close to zero for arguments outside a small interval $[-\rho, \rho]$, this means that the hypotheses of the cases are effectively satisfied more often.

3.4. Computational results. We first examine the Jacobian for a network with random initial weights on all layers and give an example. After that we present the singular values of the Jacobian and Hessian (4) for some training problems at later iterations of a Levenberg–Marquardt algorithm.

The singular value decomposition is used to measure ill-conditioning. Although the rank of J is equal to the number of its nonzero singular values, numerically it is rare to compute a singular value exactly equal to zero. To avoid the need for ad hoc numerical determinations of rank, we present all of the singular values for a given problem. Singular values and eigenvalues were computed using Matlab 3.5 running on a Sun Sparcstation-1. Jacobian and Hessian matrices were computed with 48-bit mantissa arithmetic, using a form of automatic differentiation (see [16] for implementation details).

The 2-3-2 network in Fig. 1 has 19 parameters or weights and so the Jacobian has 19 singular values. Our two-dimensional input space is sampled on a uniform mesh in $[0,1]^2$ with 400 training data points in each case presented, and a maximum of 2000 function evaluations is allowed in the Levenberg–Marquardt algorithm unless indicated otherwise. The implementation of the Levenberg–Marquardt algorithm used here was written by Moré and is available in MINPACK. Note that the initial Jacobian is *independent* of the particular problem one is solving, but *dependent* on the network and the sample distribution of the input points, t_i.

Figures 6 and 7 show the singular values of J for different sets of initial weights chosen randomly from the intervals $(-1, 1)$ and $(-100, 100)$, respectively. Other commonly used intervals differ only by scale from these two. The random seeds were kept fixed and only the weight interval was changed. The condition numbers of the Jacobian are of order 10^7 and 10^{55}, respectively. Hence the condition number of $J^T J$, used in the Newton-like methods of Table 2, is close to or smaller than the inverse of the machine epsilon for a 48-bit mantissa computer. As an example of a larger network, Fig. 8 shows the singular values of the initial Jacobian for four cases for a 5-7-2 network with weights chosen randomly in the region $(-1, 1)$ and t_i sampled randomly in the cube $[0,1]^5$.

From the graphs we can conclude that the Jacobian becomes more ill conditioned as the norm of the weight vector increases. Considering that the initial conditions of the Jacobian in Figs. 6 and 7 are not unusual (since often the initial weights are chosen randomly from these two intervals), the numerical method starts with an ill-conditioned J. In Fig. 7 the weight vectors have a larger norm on average than in the other figure of the random weights and are representative of the situation during the solution process.

Next we will investigate an extreme example for the primary sources of rank-deficiency in the Jacobian using the notation and classification given in the previous sections. We use a 2-3-2 network because

- the network is simple to visualize (Fig. 1);
- the network does not contain too many superfluous nodes, i.e., ones unnecessary for the solution of the problem and that would create additional rank-deficiency of the Jacobian;
- the network is unlikely to find a good solution since it contains too few nodes. This allows exploration of behavior far from the optimum. However, this does not mean that the Jacobian would necessarily be of full rank for an "ideal" network or a large network.

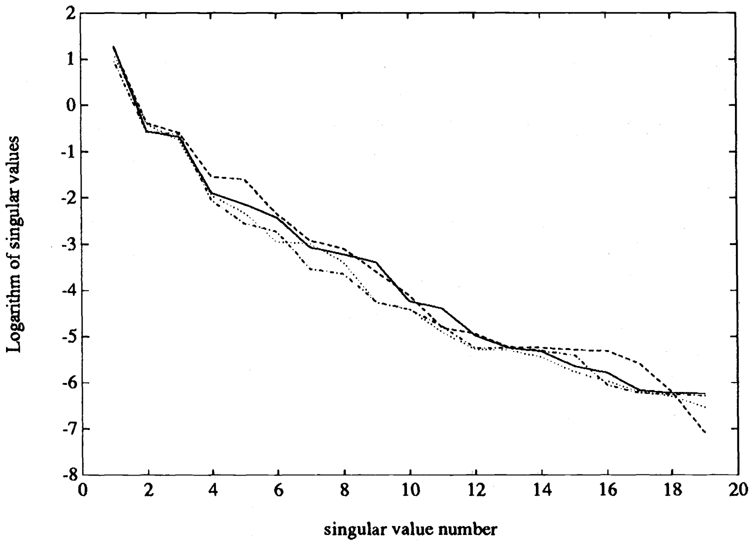


FIG. 6. *Computed singular values of four initial Jacobians for a 2-3-2 network with weights chosen randomly from* $(-1, 1)$.

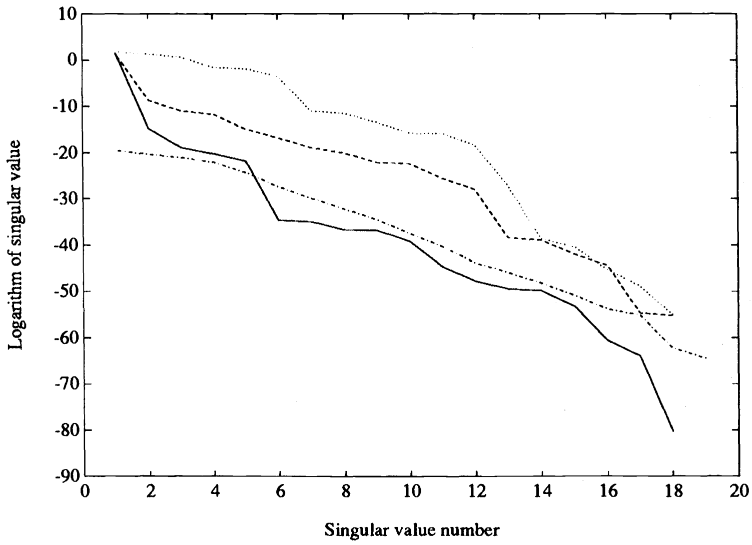


FIG. 7. *Computed singular values of four initial Jacobians for a 2-3-2 network with weights chosen randomly from* $(-100, 100)$.

As an example of the different classes of rank-deficiency, we use the initial condition in the (dash-dotted) curve in Fig. 7, where the Levenberg–Marquardt algorithm is allowed to proceed one iteration with a 2-3-2 network. The test figure is a spiral sampled on an equidistant grid as shown in Fig. 9. The value of the function inside the spiral swirls is one, and outside the swirls it is zero.

The results are summarized in Table 3. The cosines of the canonical angles between range spaces are used to measure how closely two blocks J_1 and J_2 of columns of J come to spanning the same space (as is needed for Cases 2 and 3). The method for

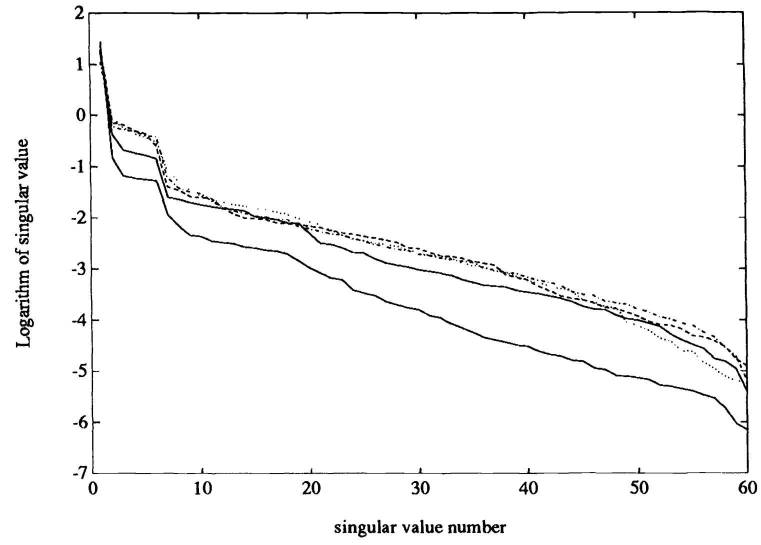


FIG. 8. *Computed singular values of five initial Jacobians for a* 5-7-2 *network.*

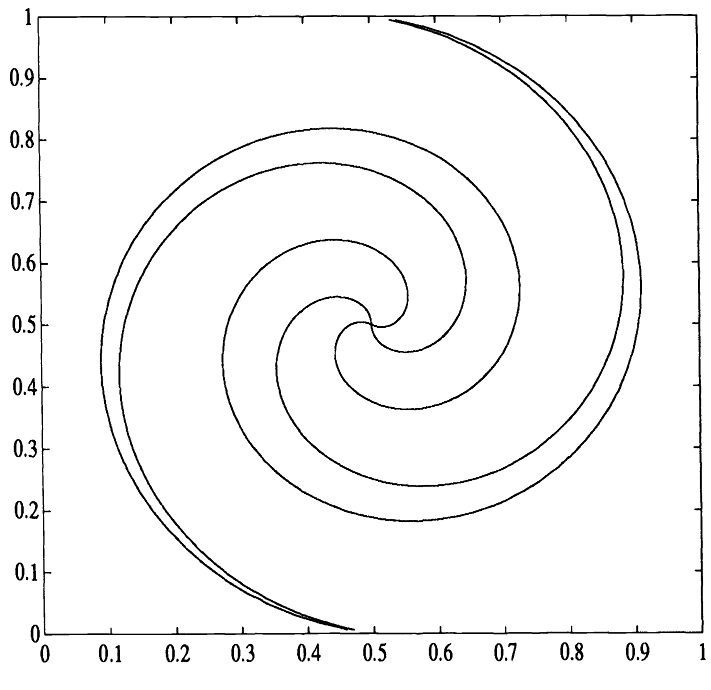


FIG. 9. *The spiral-problem. The function has value* 1 *inside the spiral and* 0 *outside.*

computing such cosines is from [1]. When some cosines are close to 1, then range(J_1) and range(J_2) are close to sharing a subspace spanned by the corresponding canonical vectors. The first column of Table 3 lists the first four dependency cases of §3.3. The second column gives the cosines of the angles between the vectors that determine rank-deficiency (for example, in Case 2 for a 2-3-2 network the determining vectors are $\underline{\sigma}'(Q_1)$ and $\underline{\sigma}'(Q_2)$), the third column specifies the columns of J with a potential resultant dependency, and the last column gives the cosine(s) of the angle(s) between the affected columns. The claims made in §3.3 are that when the angle between the determining vectors are small, then so are the angles between the affected columns as occurs for this extreme case. The singular values of columns 10–17 and columns 1–9 of the Jacobian are shown together with the singular values of the full Jacobian in Fig. 10.

TABLE 3
Example of rank deficiency.

Dependency case	Determining vectors and cosine of their angle	Affected block columns of J	Canonical cosines for affected cols.
I	e & $\underline{\sigma}(P_1)$ 0.99325	10 & 11, 14 & 15	0.99324, 0.99326
	e & $\underline{\sigma}(P_2)$ 0.99930	10 & 12, 14 & 16	0.99930, 0.99931
	e & $\underline{\sigma}(P_3)$ 0.99765	10 & 13, 14 & 17	0.99766, 0.99765
II	$\underline{\sigma}'(Q_1)$ & $\underline{\sigma}'(Q_2)$ 0.99995	[10,11,12,13] & [14,15,16,17]	1.00000, 1.00000, 0.99998, 0.99993
III	$\underline{\sigma}'(P_1)$ & $\underline{\sigma}'(P_2)$ 0.99733	[1,2,3] & [4,5,6]	1.00000, 0.99901, 0.99652
	$\underline{\sigma}'(P_1)$ & $\underline{\sigma}'(P_3)$ 0.99639	[1,2,3] & [7,8,9]	1.00000, 0.99852, 0.99503
	$\underline{\sigma}'(P_2)$ & $\underline{\sigma}'(P_3)$ 0.99861	[4,5,6] & [7,8,9]	1.00000, 0.99986, 0.99982
IV	$\underline{\sigma}(Q_1)$ & $\underline{\sigma}(Q_2)$ 0.99990	18 & 19	0.99990

Since the conditioning of network Jacobians can be represented succinctly by the singular values and the specific reason for rank-deficiency can be analyzed using the procedure above, we now show only the singular values of a few larger problems.

Figures 11 and 12 show the eigenvalues of $J^T J$ and the full Hessian for the spiral problem with a 2-14-7 network at some iterations. The initial weights are chosen randomly in the interval $[-1, 1]$. At a solution (Fig. 12) the 40-20-40 rule for a randomly chosen test set of size 1000 gives an error rate of 13.5 percent. This criterion assigns 0 to output values in [0,0.40], and 1 to values in [0.60,1]. Values in [0.40, 0.60] are labeled incorrect. Figure 13 for a 2-16-6 network shows eigenvalues at the solution (the initial weights are chosen from $[-1, 1]$). The Levenberg–Marquardt algorithm appears to be able to obtain an acceptable error norm for this problem for a large range of networks and initial conditions.

3.5. Determining cosines and rank-deficiency. We have monitored the determining and canonical cosines for some of the cases described above in a number of test problem configurations. While rank-deficiency of the Jacobian can result from interactions among columns outside of the groups analyzed separately in Cases 1–5 above, the plots show that the column groups identified above frequently exhibit in-

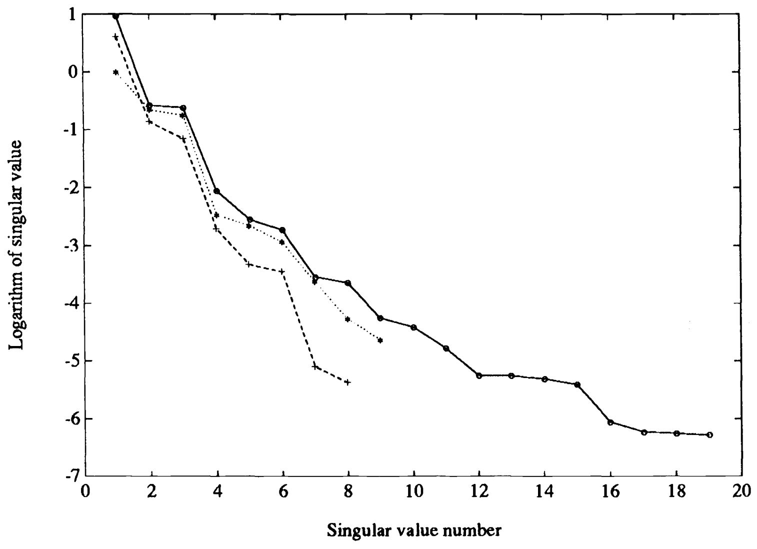


FIG. 10. *Computed singular values of columns* 10–17 (*labeled* + *with dashed line*), 1–9 (*labeled* * *with dotted line*), *and all columns* (*labeled o with solid line*) *of the Jacobian for the example discussed in the text and Table* 3 (*a* 2-3-2 *network used for the spiral problem*).

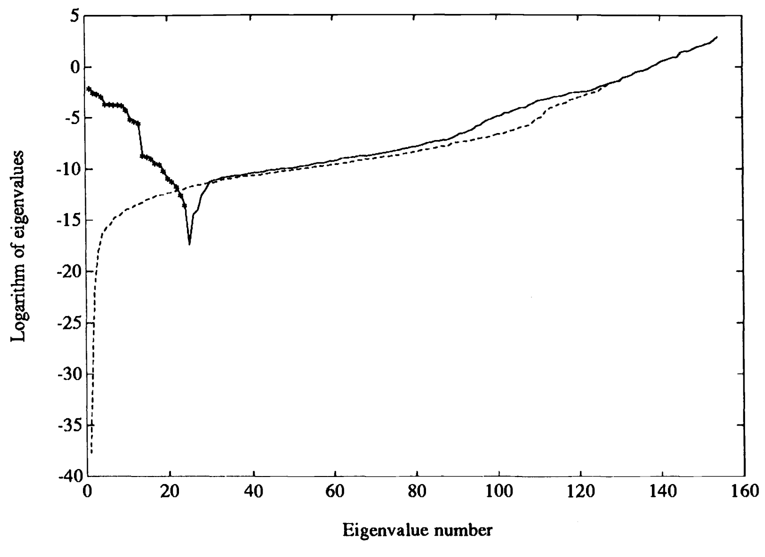


FIG. 11. *The computed eigenvalues of* $J^T J$ (*dashed line*) *and Hessian* (*solid line with* * *indicating negative eigenvalues*) *for the spiral problem with a* 2-14-7 *network at iteration* = 1000 (*the error norm was* $5.2 * 10^{-4}$).

ternal deficiency as predicted. Moreover, large determining cosines do act as sufficient conditions for rank-deficiency.

To demonstrate these facts, we have plotted the canonical and determining cosines for Case 2 in a run of the Levenberg–Marquardt solver. Those cosines as a function of the iteration number for a 2-3-2 network are shown in Fig. 14.

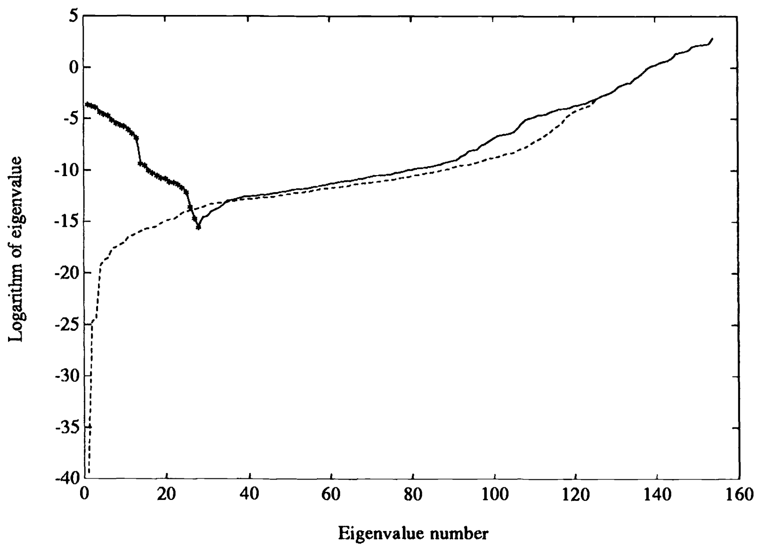


FIG. 12. *The computed eigenvalues of J^TJ (dashed line) and Hessian (solid line with * indicating negative eigenvalues) for the spiral problem (same as in the previous figure) at the solution (iteration 1832 with error norm = $4.1 * 10^{-5}$).*

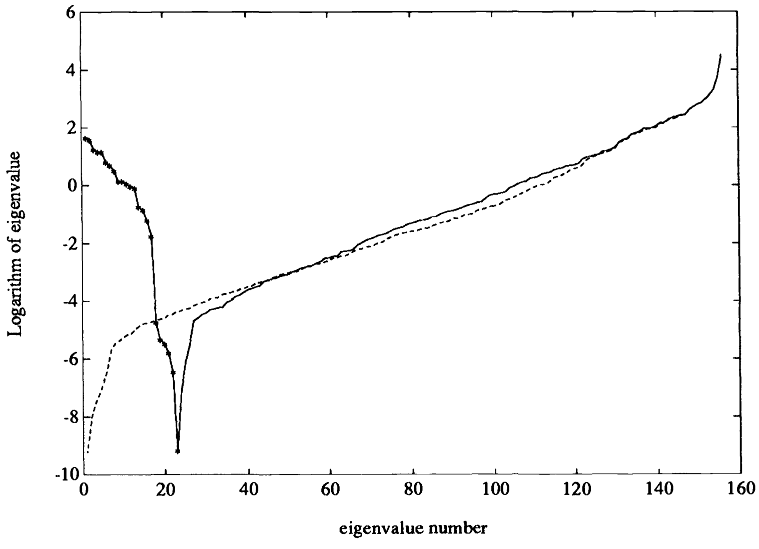


FIG. 13. *Computed eigenvalues of J^TJ (dashed line) and of the Hessian (solid line, negative eigenvalues are indicated with *) at the last iteration (error norm = 3.4 for 900 training data points and a maximum of 400 function evaluations) for the spiral problem with a 2-16-6 network. The 40-20-40 rule gives an error rate of 7.1 percent.*

Additionally, in Figs. 15 and 16 we show the canonical and determining cosines for two 2-8-4 networks, with different initial conditions, at the computed solution for each combination of columns (i.e., six pairs on the second layer). These results suggest that canonical cosines do explain rank-deficiency of the Jacobian in feedforward networks. In Fig. 16, Case 2 does not appear and the rank-deficiency comes from other causes.

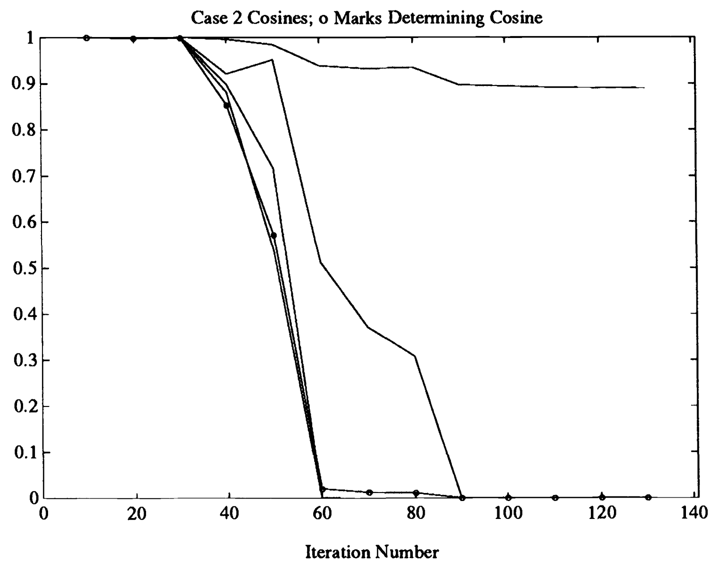


FIG. 14. *The computed cosines for Case* 2 *for a* 2-3-2 *network as a function of iteration number. The determining cosines are marked "o."*

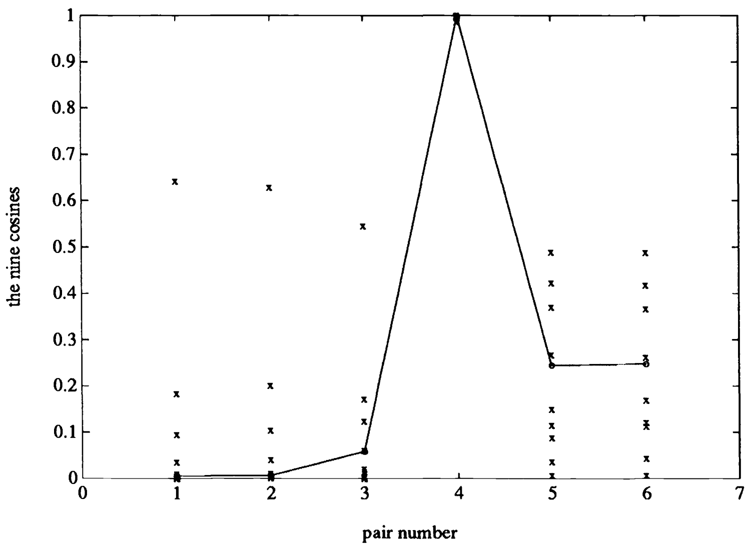


FIG. 15. *The resulting nine cosines from Case* 2 *for a* 2-8-4 *network shown versus the six pairs on the second layer. The determining cosines are indicated by "o" and connected with solid lines.*

In Figs. 17 and 18 we show the corresponding eigenvalues of the Hessian and $J^T J$ at the solution for these runs.

3.6. The Jacobian for a 2-3 network, with one hidden layer. This section presents the structure of the Jacobian for a one hidden layer network of size 2-3, that is, with two inputs and three nodes on the hidden layer. These networks have been successfully used in many neural network applications [17] and they can be used also

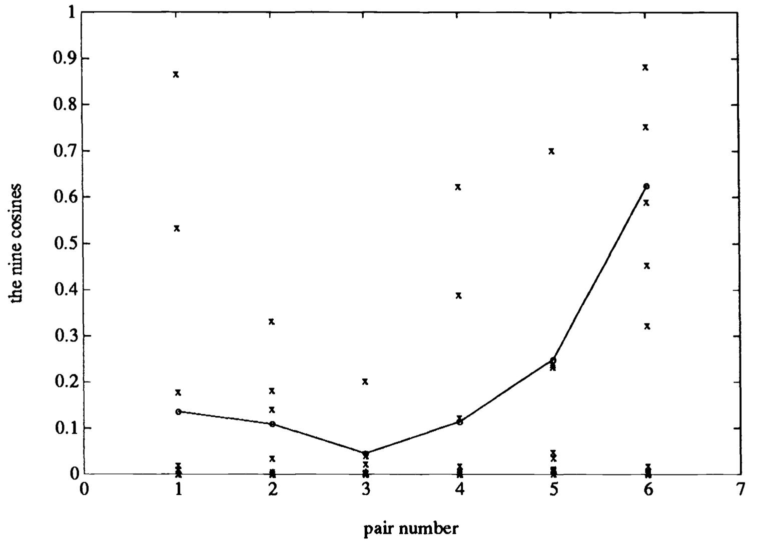


FIG. 16. *The resulting nine cosines from Case* 2 *for a* 2-8-4 *network shown versus the six pairs on the second layer. The determining cosine are indicated by "o" and connected with solid lines.*

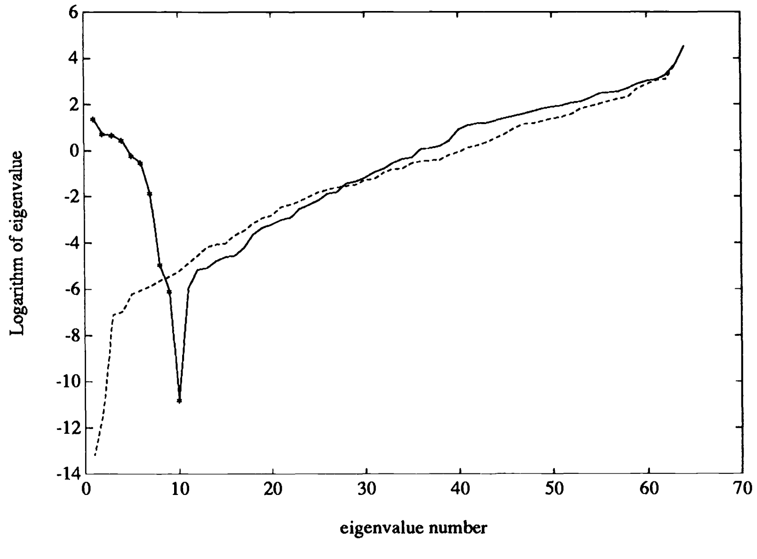


FIG. 17. *The computed eigenvalues for the Hessian (solid line with * indicating negative eigenvalues) and J^TJ (dashed line) at the solution for a* 2-8-4 *network (see also Fig.* 15). *The* 40-20-40 *rule gives the error rate* 5.5 *percent with a maximum of* 400 *iterations.*

as universal function approximators [3]. The form of the one hidden layer Jacobian is shown below with a numbering similar to that in Fig. 1.

$$\begin{aligned} J_{i,1\ldots3} &= x_{10}\sigma'(P_1^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T, \\ J_{i,4\ldots6} &= x_{11}\sigma'(P_2^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T, \\ J_{i,7\ldots9} &= x_{12}\sigma'(P_3^{(i)})(1, v_1^{(i)}, v_2^{(i)})^T, \\ J_{i,10\ldots12} &= (\sigma(P_1^{(i)}), \sigma(P_2^{(i)}), \sigma(P_3^{(i)}))^T. \end{aligned}$$

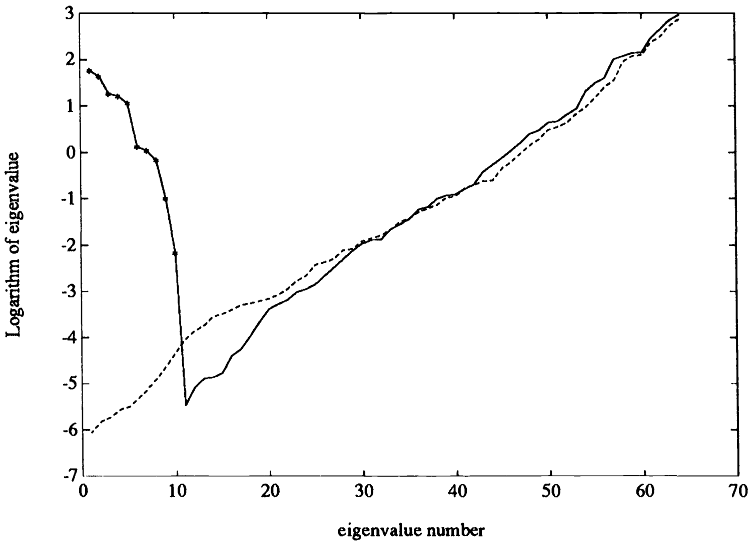


FIG. 18. *The computed eigenvalues for the Hessian (solid line with * indicating negative eigenvalues) and J^TJ (dashed line) at the solution for a* 2-8-4 *network (see also Fig.* 16). *The* 40-20-40 *rule gives the error rate* 15 *percent with a maximum of* 400 *iterations.*

The Jacobian for such a system is also likely to be rank-deficient because situations as in Case 2 appear in the equations. Columns 1–9 of the Jacobian above have the same three terms in common, i.e., the vector $(1, v_1^{(i)}, v_2^{(i)})$, which are multiplied by a term of the form $\sigma'(x)$.

4. Summary and conclusions. In a feedforward neural network the Jacobian is usually ill conditioned. Since many numerical schemes use it as a basis for their search direction, any rank-deficiency in the Jacobian causes the algorithm to obtain only partial information of the possible search directions, and in turn causes long training times. The rank-deficiency for a two hidden layer network often arises from the outputs of the second layer nodes because a sigmoidal applied to these outputs has a limited discrimination capability, especially if the weights for this level become large. Consequently, for a network with more hidden layers the rank-deficiency can only increase, but the rank-deficiency for a one hidden layer network could in principle be less severe.

Our experiments and analyses of partially connected networks and of single hidden layer networks do not indicate a significantly better conditioned Jacobian. Use of partially connected networks may inhibit rank-deficiency somewhat, but cannot guarantee complete avoidance of the intrinsic problems. A situation where some connections are omitted between two consecutive hidden layers appears in the Jacobian as the omission of some terms in the sum in (6). The apparent redundancy of a Jacobian can only be eliminated by a posteriori deletion of nodes and weights in a problem-specific way, so there would be no effect on the training process itself. Again, one could still have rank-deficient Jacobians arising in problems with either zero or nonzero residuals using reduced connectivity networks.

The Jacobian conditioning can possibly be improved by using a different form of the sigmoidal (i.e., a function with a better discrimination capability). A better initial condition might cause the Levenberg–Marquardt algorithm to find a solution with a weight vector that has a smaller norm, but practice shows that the weight vector at the solution has a large norm. The large norm of the weight vector (cf., (5)–(7)) is one reason that the algorithm causes the $\sigma'(x)$-function to approach its limit zero and therefore causes rank-deficiency in the Jacobian, as explained in §3.3.

In short, formulating feedforward neural network problems as least squares problems can cause undue strain on any numerical scheme which uses Jacobians, and a reformulation of the problem is called for.

Acknowledgments. The authors thank the anonymous referees and especially the Associate Editor, Margaret Wright, for their corrections and helpful suggestions, which have greatly improved this paper.

REFERENCES

[1] A. Björck and G. Golub, *Numerical methods for computing angles between linear subspaces*, Math. Comput., 27 (1973), pp. 579–594.

[2] M. D. Buhmann, *Multivariate interpolation in odd dimensional Euclidean spaces using multiquadratics*, Tech. Rep. DAMTP 1988/NA6, Dept. of Applied Mathematics and Theoretical Physics, Univ. of Cambridge, Cambridge, U.K., 1988.

[3] G. Cybenko, *Approximations by superpositions of a single function*, Math. Control Signals Systems, 2 (1989), pp. 303–314.

[4] J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[5] C. Fraley, *Solution of nonlinear least-squares problems*, Ph.D. thesis, Stanford Univ., Stanford, CA, June 1987; Tech. Rep. # STAN-CS-87-1165, Dept. of Computer Science, Stanford University, Stanford, CA.

[6] P. Gill, W. Murray, and M. Wright, *Practical Optimization*, Academic Press, London, New York, 1981.

[7] G. Golub and C. van Loan, *Matrix Computations*, 2nd ed., The John Hopkins University Press, Baltimore, MD, 1989.

[8] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Networks*, Addison-Wesley, Reading, MA, 1991.

[9] D. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1973.

[10] J. J. Moré, *The Levenberg–Marquardt algorithm: Implementation and theory*, in Numerical Analysis, Lecture Notes in Math., Vol. 630, G. A. Watson, ed., Springer-Verlag, Berlin, New York, 1977, pp. 105–116.

[11] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[12] M. Powell, *Radial basis functions for multivariable interpolation: A review*, in IMA Conference on Algorithms for the Approximation of Functions and Data, Oxford University Press, London, U.K., 1987.

[13] M. J. D. Powell, *Restart procedures for the conjugate gradient method*, Math. Programming, 12 (1977), pp. 241–254.

[14] ———, *Approximation theory and methods*, Cambridge University Press, London, U.K., 1981.

[15] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986.

[16] S. Saarinen, R. Bramley, and G. Cybenko, *Neural networks, backpropagation, and automatic differentiation*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. Corliss, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992, pp. 31–42.

[17] T. Sejnowski and C. Rosenberg, *Parallel networks that learn to pronounce English text*, Complex Systems, 1 (1987), pp. 1134–1142.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 715–734, May 1993

ITERATIVE LINE CUBIC SPLINE COLLOCATION METHODS FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS IN SEVERAL DIMENSIONS*

A. HADJIDIMOS†, E. N. HOUSTIS†, J. R. RICE†, AND E. A. VAVALIS†

Abstract. This paper presents a new class of second- and fourth-order line cubic spline collocation methods (the LCSC methods) for multidimensional linear elliptic partial differential equations with no cross derivative terms. The LCSC methods approximate the differential operator along lines in each dimension independently and then combine the results into one large linear system. Expressed in terms of discretization stencils for the operator, these methods have nonzero entries only in the coordinate directions. The advantage of this approach is that the discretization is much simpler to derive and analyze. Further, iterative methods are easily applied to the resulting linear systems, especially on parallel computers. The disadvantage is that the resulting linear system is k times larger in k dimensions.

Using the simplicity of the methods, iterative schemes are analyzed and formulated in order to solve the resulting LCSC linear systems in the case of Helmholtz problems. Block Jacobi, extrapolated Jacobi (EJ), and successive overrelaxation (SOR) iteration methods are analyzed with the rates of convergence and the optimum relaxation parameters determined. The simple structure of the linear system makes these methods particularly suitable for parallel computation. It is shown that the overall efficiency of the method is attractive in spite of involving such a large linear system.

Experimental results presented here confirm the convergence results for both the discretization and iterative methods, and indicate that the convergence results hold for problems more general than Helmholtz problems.

Key words. collocation methods, elliptic partial differential equations, SOR iterative method

AMS(MOS) subject classifications. 65N35, 65N05, 65F10

1. Introduction.

In this paper we formulate iterative collocation methods based on cubic spline piecewise polynomials for approximating the solution u of the second-order elliptic linear partial differential equation (PDE)

$$Lu \equiv \sum_{i=1}^{k} \alpha_i \frac{\partial^2 u}{\partial x_i^2} + \sum_{i=1}^{k} \beta_i \frac{\partial u}{\partial x_i} + \gamma u = f \quad \text{in } \Omega, \tag{1a}$$

subject to Dirichlet or Neumann boundary conditions

$$Bu = g \quad \text{on } \partial\Omega \equiv \text{ boundary of } \Omega, \tag{2a}$$

where Bu is u (or $\partial u/\partial x_i$), $\Omega \equiv \prod_{i=1}^{k} \otimes[a_i, b_i]$ is a rectangular domain in R^k (the space of k real variables), and $\alpha_i(< 0)$, β_i, $\gamma(\geq 0)$, f, and g are functions of k variables. We carry out the derivation and analysis only for the Helmholtz problem, with Dirichlet boundary conditions and constant coefficients, that is

$$Lu \equiv \sum_{i=1}^{k} \alpha_i D_{x_i}^2 u + \gamma u = f \quad \text{in } \Omega, \tag{1b}$$

$$u = g \qquad \text{on } \partial\Omega. \tag{2b}$$

*Received by the editors April 5, 1990; accepted for publication (in revised form) April 1, 1992.

†Computer Science Department, Purdue University, West Lafayette, Indiana 47907. This work was supported in part by National Science Foundation grant CCR-8619817, Air Force Office of Scientific Research grant 88-0243, and the Strategic Defense Initiative through Army Research Office contract DAAL03-86-K-0106. This paper was first presented at the First Copper Mountain Conference on Iterative Methods, Copper Mountain, Colorado, April 1–5, 1990.

The discretization is easily extended to the more general equation (1), but the analysis is not. Experimental results are given that indicate the behavior proved for this Helmholtz problem is present for more general problems.

First we formulate the cubic spline collocation discretization scheme that determines the cubic spline approximations to u along the mesh lines of a uniform partition of Ω. Throughout this paper we refer to this scheme as the *line cubic spline collocation* (LCSC) method. Following the techniques introduced in [7] and [8], we present second- and fourth-order LCSC methods for approximating the PDE problem (1b), (2b). The extension of the LCSC discretization to the more general problem (1a), (2a) is not difficult. The main objective of this paper is to define and analyze iterative linear equation solvers for computing the discrete LCSC approximation of u. For this reason, we adopt the tensor product formulation [6], [10] of the collocation equations and analyze the convergence of block Jacobi, EJ, and SOR schemes. It is worth noticing that the linear system of equations arising from the LCSC method (*the* LCSC *equations*) is nonsymmetric and lacks many of the properties found in Ritz–Galerkin-type finite element methods. Nevertheless, we derive analytic expressions for the spectral radius of the corresponding Jacobi iteration matrix, and from this we determine the convergence ranges and compute the optimal parameters for the EJ and SOR methods.

This paper is organized as follows. Section 2 includes the formulation of the second- and fourth-order LCSC schemes. Section 3 presents the formulation of first-order stationary iterative methods for the LCSC equations and the theoretical study of their convergence. The computational efficiency of these methods is analyzed also. Finally, in §4 we present the results of various numerical experiments designed for the verification of the theoretical behavior of the iterative LCSC solvers. For two Helmholtz problems in two and three dimensions, the experiments show very good agreement with the theoretical predictions about the convergence of the discretization error and the iterative method used. Although the theoretical results presented here hold for the model problem (1b), (2b), our experimental results indicate that the behavior of the iterative LCSC solvers on the general problem (1a), (2a) is similar. One of these has $\gamma = -10e^{x+y}$ in two dimensions, the other has $\alpha_1 = -1$, $\alpha_2 = -(x+z)$, $\alpha_3 = -(4-x-z)$, and $\gamma = -e^{x+y+z}$ in three dimensions.

2. The LCSC method.

2.1. Preliminary results. In this section we introduce a notation for the discrete geometric data used to define the space $S_{3,\Delta}$ of cubic splines in k dimensions and a line collocation approximation of u in $S_{3,\Delta}$. Furthermore, we adopt a representation of the coefficients of this approximation and define an interpolant S of u in $S_{3,\Delta}$ whose asymptotic behavior as a replacement of u is used to define optimal LCSC methods. The approach is to do everything uniformly in all variables so that a tensor product structure is introduced into the discretization. As is common with spline-based methods, we introduce one extra point beyond each end of the intervals $[a_i,\, b_i]$ defining the ith side of the domain Ω. This enlarged interval is then discretized uniformly with step size h_i by $\Delta_i \equiv \left\{\tau_\ell^i = a_i + \ell h_i; \ell = -1, \dots, N_i+1 \text{ with } h_i = (b_i - a_i)/N_i\right\}$. A complete discretization of Ω is obtained by taking the tensor product of these discretized lines, so the mesh $\Delta \equiv \prod_{i=1}^{k} \otimes\Delta_i$ provides an extended uniform partition of Ω. The interior mesh points are denoted by $(\tau_{n_1}^1, \tau_{n_2}^2, \dots, \tau_{n_k}^k)$, for all $\mathbf{n} \in \prod_{i=1}^{k} \otimes I_i$, where $I_i \equiv \{n_i \,:\, 1 \le n_i \le N_i - 1\}$. The set of mesh lines parallel to the x_i-axis are denoted by $\mathcal{L}^i(\mathbf{n}_i)$, where

$$\mathbf{n}_i \equiv (n_1, \ldots, n_{i-1}, n_{i+1}, \ldots, n_k)$$

indexes these $\Pi_{\ell=1}^{k}(N_\ell + 1)/(N_i + 1)$ lines. The mesh points of each set of lines $\mathcal{L}^i(\mathbf{n}_i)$ are represented by the vectors

$$\mathbf{T}_\ell^i \equiv (\tau_{n_1}^1, \ldots, \tau_{n_{i-1}}^{i-1}, \tau_\ell^i, \tau_{n_{i+1}}^{i+1}, \ldots, \tau_{n_k}^k), \quad \ell = 0, \ldots, N_i,$$

and these are used as collocation points.

The collocation approximation is made on each set $\mathcal{L}^i(\mathbf{n}_i)$ of lines using the B spline basis $\{B_\ell^i(x_i)\}_{-1}^{N_i+1}$ defined on Δ_i. We let $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ denote a variable ranging over Ω and we partition it into $\mathbf{x}^i = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)$, and x_i. The collocation approximation u_Δ^i on each line in $\mathcal{L}^i$ is represented as follows:

$$u_\Delta^i(\mathbf{x}) = \sum_{\ell=-1}^{N_i+1} U_\ell^i(\mathbf{x}^i) B_\ell^i(x_i). \tag{3}$$

Observe that (3) is the sum of one-dimensional splines in the x_i variable whose coefficients $U_\ell^i(\mathbf{x}^i)$ are functions of the other $k-1$ variables. Note that $U_\ell^i(\mathbf{x}^i)$ is an array of size $N = \Pi_i \, (N_i + 3)$, the index ℓ varies along the ith dimension, and $\mathbf{x}^i$ varies over the other $k-1$ dimensions. Further, this approximation is "redundant" in that there are k choices for representing $u_\Delta^i(\mathbf{x})$, one for each coordinate direction. This notation is illustrated in §2.3 with an example for $k = 3$. Throughout, we use the notation $U_\ell^i(\mathbf{n}_i)$ or, more simply, $U^i(\mathbf{n})$ or U^i for the coefficient values

$$U_\ell^i(\tau_{n_1}^1, \ldots, \tau_{n_{i-1}}^{i-1}, \tau_{n_{i+1}}^{i+1}, \ldots, \tau_{n_k}^k).$$

The LCSC methods considered in this paper are based on some interpolatory relations satisfied by the interpolant S of u in $S_{3,\Delta}$ defined on each line $\mathcal{L}^i(\mathbf{n}_i)$ by the interpolating equations

$$S|_{\mathbf{T}_\ell^i} = u|_{\mathbf{T}_\ell^i} \quad \text{for } \ell = 0, \ldots, N_i \tag{4}$$

and

$$D_{x_i}^2 S|_{\mathbf{T}_v^i} = D_{x_i}^2 u|_{\mathbf{T}_v^i} \quad \text{for } v = 0 \text{ and } N_i. \tag{5}$$

In (4) and (5) a particular i is used, and all of these are combined in the final collocation methods.

The approximation properties of these interpolants are summarized in the following theorem [7], [8]. The analysis of the truncation error for the fourth-order method involves an operator L', which is a perturbation of L. Specifically, we introduce $\wedge_i$, the second-order central difference operator in the ith dimension by setting $H_i = (0, \ldots, 0, h_i, 0, \ldots, 0)$ with h_i in the ith location, and then

$$\wedge_i f(\mathbf{x}) \equiv f(\mathbf{x} + H_i) - 2f(\mathbf{x}) + f(\mathbf{x} - H_i).$$

The perturbation P_L of L is then the sum over the k dimensions of the second differences of the second derivatives,

$$P_L f(\mathbf{x}) \equiv \sum_{i=1}^{k} \frac{\wedge_i}{h_i^2} D_{x_i}^2 f(\mathbf{x})$$

and

$$L' = L + P_L.$$

See [8] for a fuller explanation and motivation of this pertürbation.

THEOREM 2.1. *Suppose that* $u \in C^6[\bar{\Omega}]$, *the coefficients of* L *are in* $C[\bar{\Omega}]$ *and* $f, g \in C^4[\bar{\Omega}]$. *Then at the mesh points* $\mathbf{T}^i_\ell$ *of each line* $\mathcal{L}^i$, *we have*

$$LS = f + O(h_i^2), \qquad \ell = 0, \ldots, N_i, \tag{6}$$

and at the interior mesh points, we have

$$L'S = f + O(h^4), \qquad \ell = 1, 2, \ldots, N_i - 1. \tag{7}$$

Thus the spline S interpolant of $u(x)$ using (4) and (5) satisfies the PDEs $Lu = f$ and $L'u = f$ to second- and fourth-order, respectively. This result leads, following the analysis of [8], to the fact that the approximation $u^i_\Delta(\mathbf{x})$ defined by (3) and using the collocation points $\mathbf{T}^i_\ell$, $\ell = 0, \ldots, N_i$ has discretization error of second (for L) or fourth (for L') order.

2.2. The second-order LCSC method. In this section we present a second-order line collocation method based on cubic splines. The collocation approximation $u^i_\Delta \in S_{3,\Delta}$ is forced to satisfy (1a) at the mesh points $\{\mathbf{T}^1_l\}_{l=0}^{N_i}$ of each line $\mathcal{L}^1(\mathbf{n}_1)$ and the boundary conditions (2a) at the end nodal points $(\mathbf{T}^1_0, \mathbf{T}^1_{N_1})$. We derive the discrete equations that determine u^i_Δ for the Helmholtz equation (1b) and Dirichlet boundary conditions (2b).

$$\left.\left(\sum_{i=1}^{k} \alpha_i \, D^2_{x_i} \, u^i_\Delta + \gamma u^1_\Delta\right)\right|_{\mathbf{T}^i_\ell} = f \,|_{\mathbf{T}^i_\ell} \quad \text{for } \ell = 0, 1, \ldots, N_i. \tag{8}$$

The derivation of the corresponding collocation system for (1a), (2a), is straightforward, and our implementation is applicable to this general problem.

From the assumed representation (3) of u^i_Δ, we conclude easily that

$$D^2_{x_i} u^i_\Delta |_{T^i_\ell} = [U^i_{\ell-1}(\mathbf{n}_i) - 2U^i_\ell(\mathbf{n}_i) + U^i_{\ell+1}(\mathbf{n}_i)]/h_i^2 \tag{9}$$

and

$$u^i_\Delta |_{T^i_\ell} = [U^i_{\ell-1}(\mathbf{n}_i) + 4U^i_\ell(\mathbf{n}_i) + U^i_{\ell+1}(\mathbf{n}_i)]/6 \tag{10}$$

at the mesh (collocation) points $\mathbf{T}^i_\ell$ for $\ell = 1, 2, \ldots, N_i - 1$. First, we observe that the collocation equations obtained from the boundary conditions and differential equation at the end points of each line can be explicitly determined as follows:

$$\begin{aligned} U^i_0 &= \frac{h_i^2 f(\mathbf{T}^i_0)}{6\alpha_i}, & U^i_{-1} &= -U^i_1 + \frac{2h_i^2 f(\mathbf{T}^i_0)}{3\alpha_i}, \\ U^i_{N_i} &= \frac{h_i^2 f(\mathbf{T}^i_{N_i})}{6\alpha_i}, & U^i_{N_i+1} &= -U^i_{N_i-1} + \frac{2h_i^2 f(\mathbf{T}^i_{N_i})}{3\alpha_i}. \end{aligned} \tag{11}$$

The rest of the unknowns are determined by solving the so-called *interior collocation* equations, that is, on those lines of $\mathcal{L}^i$ not in $\partial\Omega$.

Equations (8) $(2 \le \ell \le N_i - 2)$ *away* from the boundary can be written as

$$\sum_{i=1}^{k} \frac{\alpha_i}{h_i^2}\left[U_{\ell-1}^i(\mathbf{n}_i) - 2U_\ell^i(\mathbf{n}_i) + U_{\ell+1}^i(\mathbf{n}_i)\right] + \frac{\gamma}{6}[U_{\ell-1}^1(\mathbf{n}_1) + 4U_\ell^1(\mathbf{n}_1) + U_{\ell+1}^1(\mathbf{n}_1)] = f|_{T_\ell^i}. \tag{12}$$

For lines next to the boundary, the equations have similar form with appropriately modified right sides. For example, with $i = 1$, the leftmost equations (corresponding to $x_i = a_i + h_i$ or $\ell = 1$) from the lines $\mathcal{L}^1$ have the form

$$\sum_{i=1}^{k} \frac{\alpha_i}{h_i^2}(-2U_1^i(\mathbf{n}_i) + U_2^i(\mathbf{n}_i)) + \frac{\gamma}{6}[U_0^1(\mathbf{n}_1) + 4U_1^1(\mathbf{n}_1) + U_2^1(\mathbf{n}_1)] = f(T_\ell^1). \tag{13}$$

The full matrix form of these equations is given in §3. This complex notation is illustrated next for the three-dimensional case ($k = 3$).

This discretization has redundant coefficients. There are $\Pi_i(N_i+1)$ coefficients U_ℓ^i for each i, but two are known from (11) at the boundaries so there are $K = \Pi_i(N_i - 1)$ unknowns in each collocation approximation $u_\Delta^i(\mathbf{x})$. This redundancy is handled by requiring that all these approximations agree on the mesh points, that is

$$u_\Delta^1 = u_\Delta^2 = \cdots = u_\Delta^k \quad \text{on the mesh } \Delta. \tag{14}$$

For each line, the collocation equations (12)–(14) can be written in the form

$$\sum_{i=1}^{k} \frac{a_i}{h_i^2}\mathcal{T}_{-2}\{U_\ell^i(\mathbf{n}_i)\} + \frac{\gamma}{6}\mathcal{T}_4\{U_\ell^1(\mathbf{n}_1)\} = \tilde{f}(\mathbf{T}_\ell^i),$$
$$\frac{1}{6}\mathcal{T}_4\{U_\ell^1(\mathbf{n}_1)\} - \frac{1}{6}\mathcal{T}_4\{U_\ell^i(\mathbf{n}_i)\} = \tilde{g}(\mathbf{T}_\ell^i),$$

where $\mathcal{T}_m \equiv$ tridiag $(1, m, 1)$, and $\tilde{f}$, $\tilde{g}$ are the right sides of (12)–(14) after the elimination of the predetermined boundary unknowns (11).

2.3. A three-dimensional example. We illustrate the above derivation for the case $k = 3$, where Ω is the unit cube and the operator L is $D_{x_1}^2 u + 3D_{x_2}^2 u + 5D_{x_3}^2 u - 7u = 9$. The mesh Δ has $N_1 = 4$, $N_2 = 2$, and $N_3 = 3$. This mesh is shown in Fig. 1, where the legend describes some details of the example.

The key point is to see that the various lines and associated arrays $\mathbf{T}$ and $\mathbf{U}$ are indexed by the points in the coordinate planes, plus an index along one coordinate direction. Thus $U_\ell^i(\mathbf{n}_i)$ is described as "the coefficients of the spline approximation along the $\mathcal{L}^i$ lines for $\ell = 0, 1, \ldots, N_i$." The same description applies to $\mathbf{T}_\ell^i$ with "coefficients" replaced by "collocation points."

The discretization (8) (at an interior point of the mesh) is of the form

$$D_{x_1}^2 U_\Delta^1 + 3D_{x_2}^2 U_\Delta^2 + 5D_{x_3}^2 U_\Delta^3 - 7U_\Delta^1 = f.$$

Our example grid is too coarse to have an interior point, but we see that the form of the discretization is a sum of three terms, each term along one coordinate direction as a *different representation* of the collocation approximation is used for each coordinate.

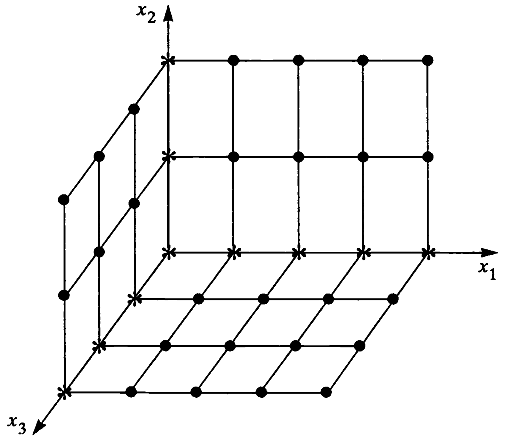


FIG. 1. *The mesh for* $k = 3$, $N_1 = 4$, $N_2 = 2$, *and* $N_3 = 3$. *The sets of lines* $\mathcal{L}^2$ *are indexed by the* 12 *points* $\mathbf{T}_0^1$ *on the plane at the left. These* 12 *points also index* $U_\ell^1(n_1) = U_\ell^1(\cdot, j_2, j_3)$ *for* $\ell = 0, \ldots, 4$, $j_2 = 0, \ldots, 2$, *and* $j_3 = 0, \ldots, 3$. *The collocation points* $\mathbf{T}_\ell^1$ *for* $\ell = 0, \ldots, 4$ *are five equally spaced points on a line starting at a* $\mathbf{T}_0^1$ *point, say* $(0, \tau_1^2 = 1/2, \tau_2^3 = 2/3)$, *and continuing to a* $\mathbf{T}_{N_1}^1$ *point, say* $(1, \tau_1^2, \tau_2^3)$.

Consider a mesh point $\mathbf{T}_j^2 = (\tau_{n_1}^1, \tau_j^2, \tau_{n_3}^3)$ on the line $\mathcal{L}^2$ (the jth point on the line indexed by $(n_1, \cdot, n_3)$). Then $D_{x_2}^2 u_\Delta^2(t)$ has three nonzero terms, namely, for those basis functions $B_j^2(x_2)$ associated with the mesh points on the (n_1, n_3) line of $\mathcal{L}^2$ having indices $j-1$, j, $j+1$. Similar considerations apply in the other two coordinates so the "stencil" for the approximation is as illustrated in Fig. 2. At the point $\mathbf{T}_j^2$, we see the seven-point star "stencil" created, i.e., the collocation equation at $\mathbf{T}_j^2$ has nonzero coefficients for the spline basis function associated with these seven points. At points next to the boundary, similar stencils are obtained with truncation as the boundary coefficients are known.

2.4. The fourth-order LCSC method. In this section, we consider the fourth-order LCSC method which uses high-order approximations of the derivatives $D_{x_i}^j u$, $j = 1, 2$. These approximations are defined as appropriate linear combinations of S and their derivatives at the mesh points [7]. Specifically, we approximate the second derivatives in the PDE operator (1b), (2b) by the difference scheme

$$\begin{aligned} D_{x_i}^2 u_\Delta^i \Big|_{\mathbf{T}_\ell^i} = {} & (U_{\ell-2}^i(\mathbf{n}_i) + 8U_{\ell-1}^i(\mathbf{n}_i) - 18U_\ell^i(\mathbf{n}_i) \\ & +8U_{\ell+1}^i(\mathbf{n}_i) + U_{\ell+1}^i(\mathbf{n}_i))/(12h_i^2). \end{aligned} \tag{15}$$

The collocation equations corresponding to the mesh points on a line $\mathcal{L}^i$ *away* $(2 \leq \ell \leq N_i - 2)$ from the boundary are written at the point T_ℓ^i as

$$\begin{aligned} \sum_{i=1}^{k} \frac{\alpha_i}{12h_i^2} & \left[U_{\ell-2}^i(\mathbf{n}_i) + 8U_{\ell-1}^i(\mathbf{n}_i) - 18U_\ell^i(\mathbf{n}_i) + 8U_{\ell+1}^i(\mathbf{n}_i) + U_{\ell+2}^i(\mathbf{n}_i)\right] \\ & + \frac{\gamma}{6}[U_{\ell-1}^1(\mathbf{n}_1) + 4U_\ell^1(\mathbf{n}_1) + U_{\ell+1}^1(\mathbf{n}_1)] = f|_{\mathbf{T}_\ell^i}. \end{aligned} \tag{16}$$

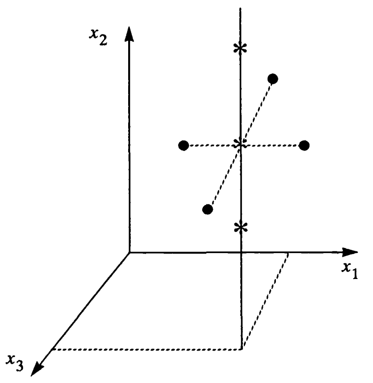


FIG. 2. *The "stencil" at an interior created by the* LCSC *approximation of* (8). *A line in $\mathcal{L}^2$ is shown with index (n_1, n_3). The three* x *mesh points on this line correspond to basis elements $B_j^2(x_2)$ which are nonzero at $\mathbf{T}_j^2$. The solid dots in the x_1 and x_2 directions from $\mathbf{T}_j^2$ are similar mesh points on other lines through $\mathbf{T}_j^2$.*

For lines *close* to the boundary, we have similar forms with appropriately modified right sides. More specifically, in the case of line $\mathcal{L}^1(2,1,\ldots,1)$, the corresponding equations have the form

$$\sum_{i=1}^{k} \frac{\alpha_i}{12h_i^2}[8U_{\ell-1}^i(\mathbf{n}_i) - 18U_\ell^i(\mathbf{n}_i) + 8U_{\ell+1}^i(\mathbf{n}_i) + U_{\ell+2}^i(\mathbf{n}_i)] + \frac{\gamma}{6}[U_{\ell-1}^1(\mathbf{n}_1) + 4U_\ell^1(\mathbf{n}_1) + U_{\ell+1}^1(\mathbf{n}_1)] = f|_{\mathbf{T}_\ell^i}. \tag{17}$$

This discretization has redundant coefficients just as the second-order LCSC method does. The same conditions (14) are used to reduce the number of coefficients to the number of equations. The nature of this discretization is changed from the previous one in that the stencils have five, rather than three, points along each coordinate direction.

3. Iterative solution of the interior LCSC equations.

In this section we consider the iterative solution of the interior collocation equations corresponding to the second- and fourth-order LCSC methods. For the matrix formulation of these equations, we introduce the notation $\mathcal{T}_m \equiv$ tridiag $(1, m, 1)$ and I for the identity matrix. Furthermore, we assume that the ordering for both the collocation equations and the unknowns in the vector $\mathbf{U}^i = \left\{U^i(\mathbf{n}), \mathbf{n} \in \prod_{i=1}^k \otimes I_i\right\}$ is the natural one. Following the above conventions and notations, the LCSC equations can be written in the form

$$\sum_{i=1}^{k} A_i \mathbf{U}^i(\mathbf{n}) = \mathbf{F}^1, \tag{18}$$

where the coefficient matrices are defined by

$$A_i \equiv \left(\prod_{j=1}^{k-i} \otimes I\right) \otimes \mathcal{E}_i \otimes \left(\prod_{j=k-i+2}^{k} \otimes I\right), \qquad i = 1, \ldots, k, \tag{19}$$

with

$$\mathcal{E}_1 = \frac{\alpha_1}{h_1^2}\mathcal{E} + \frac{\gamma}{6}\mathcal{T}_4 \quad \text{and} \quad \mathcal{E}_i = \frac{\alpha_i}{h_i^2}\mathcal{E}, \qquad i = 2,\ldots,k. \tag{20}$$

The matrix $\mathcal{E}$ depends on the discretization scheme applied. Specifically, $\mathcal{E} = \mathcal{T}_{-2}$ for the second-order discretization scheme and $\mathcal{E} = \frac{1}{12}\mathcal{T}_{10}\mathcal{T}_{-2}$ for the fourth-order one. In both cases, the system (18) is underdetermined with $\mathcal{K} = \prod_{j=1}^{k}(N_j - 1)$ equations and $k\mathcal{K}$ unknowns. For its completion, we consider the $k-1$ interpolation equations (14). If we order them according to the ordering of unknowns $\mathbf{U}^1(\mathbf{n})$, we obtain the equations

$$-B_i\mathbf{U}^1(\mathbf{n}) + D_i\mathbf{U}^i(\mathbf{n}) = \mathbf{F}^i, \qquad i = 2,\ldots,k, \tag{21}$$

where

$$B_i = \left(\prod_{j=1}^{k-1} \otimes I\right) \otimes \mathcal{T}_4, \qquad D_i = \left(\prod_{j=1}^{k-i} \otimes I\right) \otimes \mathcal{T}_4 \otimes \left(\prod_{j=k-i+2}^{k} \otimes I\right), \tag{22}$$
$$i = 2,\ldots,k.$$

Most of the components of $\mathbf{F}^i$ are zero, except those that include the effect of the elimination of the boundary conditions. Collecting together the interior equations (18) and the auxiliary interpolation equations (21), we obtain the order $k\mathcal{K}$ system of linear equations

$$\begin{pmatrix} A_1 & A_2 & A_3 & \ldots & A_k \\ -B_2 & D_2 & & & \\ -B_3 & & D_3 & & \\ \vdots & & & \ddots & \\ -B_k & & & & D_k \end{pmatrix} \begin{pmatrix} \mathbf{U}^1 \\ \mathbf{U}^2 \\ \mathbf{U}^3 \\ \vdots \\ \mathbf{U}^k \end{pmatrix} = \begin{pmatrix} \mathbf{F}^1 \\ \mathbf{F}^2 \\ \mathbf{F}^3 \\ \vdots \\ \mathbf{F}^k \end{pmatrix} \tag{23}$$

for the unknown $\mathbf{U}^i$.

3.1. Iterative solution of the LCSC equations. Various iterative solution methods for the LCSC equations (23) for two-dimensional problems have been considered in [7] and [8]. In this paper we formulate and analyze the block Jacobi, EJ, and SOR iteration scheme for the k-dimensional problems (1b), (2b). For the convergence analysis of these iterative schemes, we denote by $\rho(K)$ the spectral radius of a matrix K, ω either the extrapolation or the relaxation SOR parameter, and J, J_ω, and $\mathcal{L}_\omega$ the Jacobi, EJ, and the SOR iteration matrices, respectively.

Next we state two very useful lemmas for the analysis that follows. The first lemma is stated in [13, pp. 107–109], but the proof given here is more general.

LEMMA 3.1. *Let the $n\times n$ matrices $A_{i,j}, i,j = 1,\ldots,k$, possess a common linearly independent set of n eigenvectors $\mathbf{y}^{(\ell)}, \ell = 1,\ldots,n$, with corresponding eigenvalues $\lambda_{i,j}^{(\ell)}$. Then the eigenvalues of the matrix*

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \ldots & A_{1,k} \\ A_{2,1} & A_{2,2} & \ldots & A_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ A_{1,k} & A_{k,2} & \ldots & A_{k,k} \end{pmatrix} \tag{24}$$

are the eigenvalues of the matrices

$$\Lambda_\ell = \begin{pmatrix} \lambda_{1,1}^{(\ell)} & \lambda_{1,2}^{(\ell)} & \cdots & \lambda_{1,k}^{(\ell)} \\ \lambda_{2,1}^{(\ell)} & \lambda_{2,2}^{(\ell)} & \cdots & \lambda_{2,k}^{(\ell)} \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_{1,k}^{(\ell)} & \lambda_{k,2}^{(\ell)} & \cdots & \lambda_{k,k}^{(\ell)} \end{pmatrix}, \qquad \ell = 1, \dots, n. \tag{25}$$

$\sigma(A) = \cup\sigma(\Lambda_\ell)$, *where* $\sigma(K)$ *denotes the spectrum of a matrix* K.

Proof. First we consider the $n \times n$ matrix $Y = [\mathbf{y}^{(1)}\mathbf{y}^{(2)} \dots \mathbf{y}^{(n)}]$ whose columns are the linearly independent eigenvectors $\mathbf{y}^{(\ell)}, \ell = 1, \dots, n$, and the diagonal matrices $\Lambda_{i,j}, i, j = 1, \dots, n$, with diagonal elements the eigenvalues $\lambda_{i,j}^{(\ell)}$ of the matrices $A_{i,j}, i, j = 1, \dots, k$. From the invertibility of Y, we conclude that

$$A_{i,j} = Y\Lambda_{i,j}Y^{-1}, \qquad i, j = 1, \dots, k. \tag{26}$$

Furthermore, A can be written in the form

$$A = (I \otimes Y)\Lambda(I \otimes Y)^{-1}, \tag{27}$$

with

$$\Lambda = \begin{pmatrix} \Lambda_{1,1} & \Lambda_{1,2} & \cdots & \Lambda_{1,k} \\ \Lambda_{2,1} & \Lambda_{2,2} & \cdots & \Lambda_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ \Lambda_{k,1} & \Lambda_{k,2} & \cdots & \Lambda_{k,k} \end{pmatrix}. \tag{28}$$

This implies that A and Λ are similar matrices. Thus it is sufficient to show the similarity of Λ and of diag $(\Lambda_1, \dots, \Lambda_n)$ with $\Lambda_\ell, \ell = 1, \dots, n$, being given in (25). For this purpose, we construct the graph $G(\Lambda)$ of Λ (see Fig. 3), where only the diagonal elements of the submatrices $\Lambda_{i,j}$ are considered, including the zero ones. In view of $G(\Lambda)$, we construct the permutation matrix P such that $p_{i,j} = 1$ for (i, j) in the set

$$\begin{gathered}\{(1,1), (2, n+1), \dots, (n-1, (k-2)n+1), (n, (k-1)n+1), \\ (n+1, 2), (n+2, n+2), \dots, (2n-1, (k-2)n+2), (2n, (k-1)n+2), \\ \dots \\ ((k-1)n+1, n), ((k-1)n+2, 2n), \dots, (kn-1, (k-1)n), (kn, kn)\}.\end{gathered}$$

From the construction of P, we have $P\Lambda P^T = \text{diag}\,(\Lambda_1, \dots, \Lambda_n)$. This completes the proof of the lemma. $\square$

The second lemma to be used later for the eigenvalue analysis of the collocation coefficient matrix problem is stated as follows.

LEMMA 3.2. *Let the* $n \times n$ *matrices* $A_i, i = 1, \dots, k$, *possess a common linearly independent set of eigenvectors* $\mathbf{y}^{(j)}, j = 1, \dots, n$, *with corresponding eigenvalues* $\lambda_i^{(j)}$ $(i = 1, \dots, k, j = 1, \dots, n)$. *Then the matrix* $A \equiv A_1 \otimes A_2 \otimes \cdots \otimes A_k$ *possesses the* n^k *linearly independent eigenvectors* $\mathbf{y}^{(\underline{j})} \equiv \mathbf{y}^{(j_1)} \otimes \mathbf{y}^{(j_2)} \otimes \cdots \otimes \mathbf{y}^{(j_k)}$, *where* $\underline{j} \equiv (j_1, \dots, j_k)$ *and* $j_i = 1, \dots, n$, $i = 1, \dots, k$, *and eigenvalues* $\lambda^{(\underline{j})} = \lambda_1^{(j_1)}\lambda_2^{(j_2)} \dots \lambda_k^{(j_k)}$.

Proof. First, using tensor product properties, we can easily verify that $A\mathbf{y}^{(\underline{j})} = \lambda^{(\underline{j})}\mathbf{y}^{(\underline{j})}$. In order to prove the linear independence of the $\mathbf{y}^{(\underline{j})}$'s, we construct the

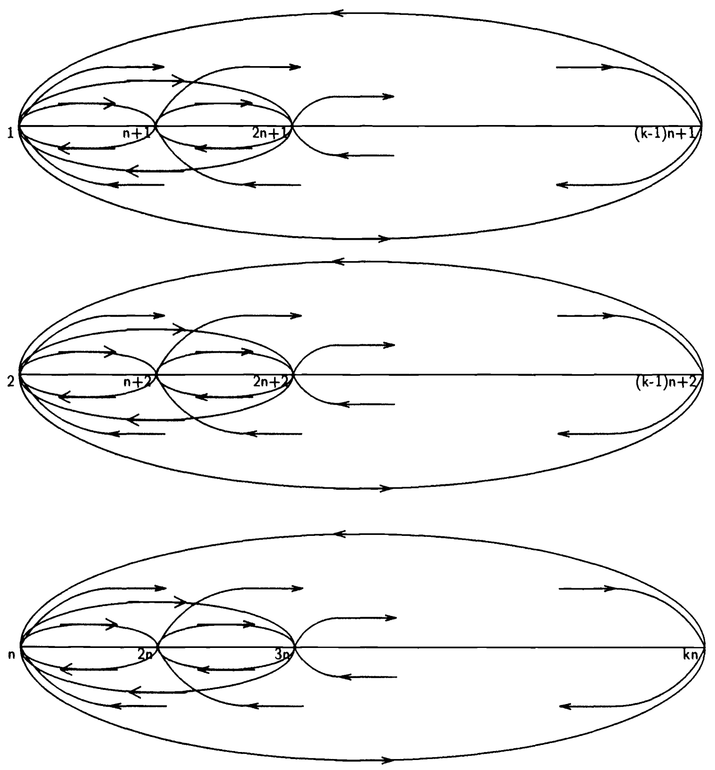


FIG. 3. *Graph $G(\Lambda)$ of the matrix Λ.*

$(n^k) \times (n^k)$ matrix Z whose columns are the n^k eigenvectors of A in the following order

$$\begin{aligned} Z = [&\mathbf{y}^{(1)} \otimes \mathbf{y}^{(1)} \otimes \ldots \otimes \mathbf{y}^{(1)}, \mathbf{y}^{(1)} \otimes \mathbf{y}^{(1)} \otimes \ldots \otimes \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(1)} \otimes \mathbf{y}^{(1)} \otimes \ldots \otimes \mathbf{y}^{(n)}, \\ &\mathbf{y}^{(1)} \otimes \mathbf{y}^{(1)} \otimes \ldots \otimes \mathbf{y}^{(2)} \otimes \mathbf{y}^{(1)}, \mathbf{y}^{(1)} \otimes \mathbf{y}^{(1)} \otimes \ldots \otimes \mathbf{y}^{(2)} \otimes \mathbf{y}^{(2)}, \ldots, \\ &\mathbf{y}^{(1)} \otimes \mathbf{y}^{(1)} \otimes \ldots \otimes \mathbf{y}^{(2)} \otimes \mathbf{y}^{(n)}, \ldots, \mathbf{y}^{(n)} \otimes \mathbf{y}^{(n)} \otimes \ldots \otimes \mathbf{y}^{(n)}]. \end{aligned}$$

Applying the tensor product notation, we obtain

$$Z = Y \otimes Y \otimes \cdots \otimes Y, \tag{29}$$

where $Y = [\mathbf{y}^{(1)}\mathbf{y}^{(2)} \ldots \mathbf{y}^{(n)}]$. From the assumed linear independence of $\mathbf{y}^{(j)}$, $j = 1, \ldots, n$, we conclude that Y^{-1} and $Z^{-1} = Y^{-1} \otimes Y^{-1} \otimes \cdots \otimes Y^{-1}$ exist. This implies the linear independence of the n^k eigenvectors and concludes the proof of the lemma. □

For the formulation of the block iterative solutions to the LCSC system (23), we consider the splitting $K - L - M$ of the coefficient matrix with $K \equiv \text{diag}(A_1, D_2, D_3, \ldots, D_k)$, $L \equiv \{L_{i,j}\}$, and $M \equiv \{M_{i,j}\}$, where the entries of the matrices L and

M are defined such that

$$
(30)\qquad \begin{aligned} L_{i,j} &\equiv \begin{cases} B_i & \text{if } j=1 \quad \text{and} \quad i=2,\dots,k, \\ 0 & \text{otherwise,} \end{cases} \\ M_{i,j} &\equiv \begin{cases} -A_i & \text{if } i=1 \quad \text{and} \quad j=2,\dots,k, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}
$$

According to the above splitting, the block Jacobi, EJ, and SOR iteration equations of (23) are defined as follows:

$$
(31)\qquad K\mathbf{U}^{(s+1)} = (L+M)\mathbf{U}^{(s)} + \mathbf{F}, \qquad s=0,1,2,\dots,
$$

$$
(32)\qquad K\mathbf{U}^{(s+1)} = (1-\omega)K\mathbf{U}^{(s)} + \omega(L+M)\mathbf{U}^{(s)} + \omega\mathbf{F}, \qquad s=0,1,2,\dots,
$$

and

$$
(33)\qquad \begin{aligned} K\mathbf{U}^{(s+1)} = (1-\omega)K\mathbf{U}^{(s)} + \omega L\mathbf{U}^{(s+1)} + \omega M\mathbf{U}^{(s)} + \omega\mathbf{F}, \\ s=0,1,2,\dots. \end{aligned}
$$

THEOREM 3.1. *The eigenvalues μ of the block Jacobi iteration matrix $K^{-1}(L+M)$ defined in* (31) *are $\mu=0$ of multiplicity $(k-2)\prod_{i=1}^{k}(N_i-1)$ and pure imaginary given for the second-order collocation scheme by*

$$
(34)\qquad \mu := \mu^{(\underline{\ell})} = \pm i\sqrt{\frac{\sum_{j=2}^{k}\frac{\alpha_j}{h_j^2}\lambda_j^{(\ell_j)}(6+\lambda_j^{(\ell_j)})^{-1}}{\left(\frac{\alpha_1}{h_1^2}\lambda_1^{(\ell_1)}+\beta\right)(6+\lambda_1^{(\ell_1)})^{-1}}},
$$

while for the fourth-order scheme by

$$
(35)\qquad \mu := \mu^{(\underline{\ell})} = \pm i\sqrt{\frac{\sum_{j=2}^{k}\frac{\alpha_j}{h_j^2}\lambda_j^{(\ell_j)}(12+\lambda_j^{(\ell_j)})(6+\lambda_j^{(\ell_j)})^{-1}}{\left(\frac{\alpha_1}{h_1^2}\lambda_1^{(\ell_1)}(12+\lambda_1^{(\ell_1)})+12\beta\right)(6+\lambda_1^{(\ell_1)})^{-1}}},
$$

where $i=\sqrt{-1}$, $\underline{\ell}=(\ell_1,\dots,\ell_k), \lambda_j^{(\ell_j)} = -4\sin^2(\ell_j\pi/2N_j), \ell_j=1,\dots,N_j-1, j=1,\dots,k$, and $\beta=\frac{\gamma}{6}(6+\lambda_1^{(\ell_1)})$.

Proof. We start by pointing out that $\mathcal{T}_{10}=12I+\mathcal{T}_{-2}$ and $\mathcal{T}_4=6I+\mathcal{T}_{-2}$. Hence the matrices $\mathcal{T}_{10},\mathcal{T}_4$, and $\mathcal{T}_{-2}$ have a common linearly independent set of eigenvectors. Furthermore, in view of (19), (20), and (22), the matrices A_i, B_i, and D_i also have a common linearly independent set of $\mathcal{K}=\prod_{i=1}^{k}(N_i-1)$ eigenvectors. The block Jacobi iteration matrix $J\equiv K^{-1}(L+M)=\{J_{i,j}\}$, associated with the matrix coefficient in (23), can be described as

$$
(36)\qquad \begin{aligned} &J_{1,1}=0, \qquad \left\{\{J_{i,j}=0\}_{j=2}^{k}\right\}_{i=2}^{k}, \\ &\{J_{1,j}=-A_1^{-1}A_j\}_{j=2}^{k}, \qquad \{J_{i,1}=D_i^{-1}B_i\}_{i=2}^{k}. \end{aligned}
$$

Consequently, the $J_{i,j}$'s possess the same $\mathcal{K}$ linearly independent eigenvectors. So, if we denote by $\lambda_{J_{i,j}}^{(\underline{\ell})}$ the eigenvalue of the matrix $J_{i,j}$ associated with the $\underline{\ell}$ common eigenvector, and use Lemma 3.1, we conclude that

$$\det \begin{pmatrix} -\mu & \lambda_{H_2}^{(\underline{\ell})} & \lambda_{H_3}^{(\underline{\ell})} & \cdots & \lambda_{H_k}^{(\underline{\ell})} \\ \lambda_{G_2}^{(\underline{\ell})} & -\mu & & & \\ \lambda_{G_3}^{(\underline{\ell})} & & -\mu & & \\ \vdots & & & \ddots & \\ \lambda_{G_k}^{(\underline{\ell})} & & & & -\mu \end{pmatrix} = 0, \tag{37}$$

where $\lambda_{G_i}^{(\underline{\ell})}$ and $\lambda_{H_i}^{(\underline{\ell})}$ are eigenvalues of the matrices $G_i := D_i^{-1}B_i$ and $H_i := -A_1^{-1}A_i$, respectively, corresponding to the $\underline{\ell}$ common eigenvector. Using Lemma 3.2, (37) gives

$$(-\mu)^k - \lambda_{G_2}^{(\underline{\ell})}\lambda_{H_2}^{(\underline{\ell})}(-\mu)^{k-2} - \cdots - \lambda_{G_k}^{(\underline{\ell})}\lambda_{H_k}^{(\underline{\ell})}(-\mu)^{k-2} = 0$$

or

$$(-\mu)^{k-2}\left[\mu^2 - \sum_{i=2}^{k} \lambda_{G_i}^{(\underline{\ell})}\lambda_{H_i}^{(\underline{\ell})}\right] = 0.$$

We conclude the proof of the theorem by observing that the eigenvalues of the $N \times N$ tridiagonal matrix $\mathcal{T}_{-2}$ are given by $-4\sin^2(\ell\pi/2(N+1)), \ell = 1, \ldots, N$. (See [14], [15], or [13].) □

The spectral radius $\rho(J)$ of the block Jacobi matrix of the iteration method (31) is obviously the largest modulus of $\mu^{(\underline{\ell})}$ given in (34) and in (35) for the second- and fourth-order collocation schemes, respectively. Explicit formulas have been found for $\rho(J)$, which are summarized in the following theorem.

THEOREM 3.2. *Let us denote by*

$$y_j(\lambda) := \left[\frac{\alpha_j}{h_j^2}\lambda + \gamma_j(6+\lambda)\right](6+\lambda)^{-1} \tag{38}$$

and by

$$z_j(\lambda) := \left[\frac{\alpha_j}{h_j^2}\lambda(12+\lambda) + 12\gamma_j(6+\lambda)\right](6+\lambda)^{-1}, \tag{39}$$

with $\lambda = \lambda_j^{(\ell_j)}$, $\gamma_1 = \frac{\gamma}{6}$, *and* $\gamma_j = 0, j = 2, \ldots, k$, *the expressions appearing in* (34) *and* (35), *respectively, and let* $s_j := -4\sin^2(\pi/(2N_j))$ *and* $c_j := -4\cos^2(\pi/(2N_j)), j = 1, \ldots, k$. *Then the spectral radii of the block Jacobi matrices corresponding to the second- and fourth-order collocation schemes are defined by the following expressions*:

$$\rho^2(J) \equiv \sum_{j=2}^{k} y_j(c_j)/y_1(s_1) \tag{40}$$

and

$$\rho^2(J) \equiv \sum_{j=2}^{k} z_j(c_j)/z_1(s_1). \tag{41}$$

Proof. First we note that in (34) and (35), $\alpha_j < 0$, $\lambda_j^{(\ell_j)} \in [c_j, s_j]$, and $\gamma_j \geq 0, j = 1, \ldots, k$. Hence the expressions $y_j := y_j(\lambda_j^{(\ell_j)}), j = 1, \ldots, k$, defined in (38) are positive and independent of each other. So are the z_j's ($:= z_j(\lambda_j^{(\ell_j)})$'s, $j = 1, \ldots, k$) defined in (39). Therefore, in order to determine the two spectral radii, it suffices to determine the maximum values of $y_j, z_j, j = 2, \ldots, k$, and the minimum values of y_1, z_1. For the extreme values of $y_j := y_j(\lambda), \lambda \in [c_j, s_j], j = 1, \ldots, k$, we differentiate y_j with respect to λ to obtain

$$\frac{\partial y_j}{\partial \lambda} = \frac{1}{(6+\lambda)^2} \frac{\alpha_j}{h_j^2} < 0.$$

Consequently,

$$\max y_j = y_j(c_j), \qquad \min y_j = y_j(s_j). \tag{42}$$

To determine the extreme values of $z_j := z_j(\lambda), j = 1, \ldots, k$, we work in a similar way. This time we have

$$\frac{\partial z_j}{\partial \lambda} = \frac{1}{(6+\lambda)^2} \frac{\alpha_j}{h_j^2} \left[(6+\lambda)^2 + 36\right] < 0.$$

This gives

$$\max z_j = z_j(c_j), \qquad \min z_j = z_j(s_j). \tag{43}$$

From the expressions (34), (35), (38), and (39), and the results in (42) and (43), the expressions (40) and (41) for the spectral radii readily follow. □

Remark. From the above theorem, it is easy to conclude that the *block Jacobi* iterative method defined by (31) does not always converge. Furthermore, the optimum *block extrapolated* SOR method is not an improvement compared to the optimum *block* SOR method (see [1], [11], and [5]).

Next we state two well-known theorems which in conjunction with (40) and (41) allow us to determine the convergence ranges and compute the optimal parameters in the case of EJ and SOR methods when the spectrum of the Jacobi matrix is purely imaginary. For their proofs, refer to [2]–[5], and [9], [12], [11], and [5], respectively.

THEOREM 3.3. *Let A be the coefficient matrix of a linear system in block partitioning form with square nonsingular diagonal blocks and J the corresponding Jacobi matrix. If all the eigenvalues of J^2 are nonpositive, then, for the convergence of the block EJ method, we have*

$$\rho(J_\omega) < 1 \quad \text{if } 0 < \omega < \frac{2}{\sqrt{1+\rho^2(J)}[\rho(J) + \sqrt{1+\rho^2(J)}]}, \tag{44}$$

$$\omega_{opt} = \frac{2}{1+\rho^2(J)} \quad \text{and} \quad \rho(J_{\omega_{opt}}) = \frac{\rho(J)}{\sqrt{1+\rho^2(J)}}. \tag{45}$$

THEOREM 3.4. *If, in addition to the assumptions of Theorem 3.3, A is block 2-cyclic consistently ordered, then, for the convergence and the optimal convergence of the block SOR method, we have the following conditions:*

$$\rho(\mathcal{L}_\omega) < 1 \quad \text{if } 0 < \omega < \frac{2}{1+\rho(J)}, \tag{46}$$

$$\omega_{opt} = \frac{2}{1+\sqrt{1+\rho^2(J)}} \quad \textit{and} \quad \rho(\mathcal{L}_{\omega_{opt}}) = 1 - \omega_{opt}. \tag{47}$$

A straightforward comparison of the optimal spectral radii in (45) and (47) leads to the following theorem.

THEOREM 3.5. *Under the assumptions of Theorems* 3.2, 3.3, *and* 3.4 *the optimal* SOR *method converges asymptotically faster than the optimal* EJ *method.*

For the efficiency of both the serial and parallel iterative solution of the collocation equations we reorder the system (23) following

(a) the *cyclic natural* ordering for the unknowns $U^i, i = 2, \dots, k$,

$$\mathcal{U}^i \equiv \left\{ \cdots \left\{ \left\{ \cdots \left\{ U^i_{j_1,\dots,j_{i-1},j_i,\dots,j_k} \right\}_{j_i=1}^{N_i-1} \cdots \right\}_{j_k=1}^{N_k-1} \right\}_{j_1=1}^{N_1-1} \cdots \right\}_{j_{i-1}=1}^{N_{i-1}-1} ; \tag{48}$$

(b) the ordering of (18) according to the ordering of $\mathcal{U}^1$; and

(c) the ordering of each block of auxiliary conditions (21) according to the ordering of the unknowns $\mathcal{U}^i$.

Assume that the matrices P_i are the permutation matrices that reorder the unknowns U^i from the *natural* ordering to the *cyclic natural* ordering (48), namely, $\mathcal{U}^i = P_i \mathbf{U}^i, i = 1, \dots, k$. In view of this transformation, the new form of (23) is

$$\begin{pmatrix} A_1 & A_2P_2^T & A_3P_3^T & \dots & A_kP_k^T \\ -P_2B_2 & P_2D_2P_2^T & & & \\ -P_3B_3 & & P_3D_3P_3^T & & \\ \vdots & & & \ddots & \\ -P_kB_k & & & & P_kD_kP_k^T \end{pmatrix} \begin{pmatrix} \mathbf{U}^1 \\ P_2\mathbf{U}^2 \\ P_3\mathbf{U}^3 \\ \vdots \\ P_k\mathbf{U}^k \end{pmatrix} = \begin{pmatrix} \mathbf{F}^1 \\ P_2\mathbf{F}^2 \\ P_3\mathbf{F}^3 \\ \vdots \\ P_k\mathbf{F}^k \end{pmatrix}. \tag{49}$$

The new structure of the collocation coefficient matrix for the second- and fourth-order scheme in two dimensions is given in Fig. 4.

THEOREM 3.6. *Let A and $\mathcal{A}$ be the coefficient matrices in* (23) *and* (49), *respectively, D and $\mathcal{D}$ their block diagonal parts, and J and $\mathcal{J}$ the associated block Jacobi matrices. We have that $\mathcal{A} = PAP^T$ and $\mathcal{J} = PJP^T$, where $P = (I, P_2, \dots, P_k)$, and therefore $\mathcal{A}$ and A as well as $\mathcal{J}$ and J are similar matrices.*

Proof. The relation $\mathcal{A} = PAP^T$ follows easily from the formulation of the method. For the relation $\mathcal{J} = PJP^T$, we have

$$\begin{aligned} \mathcal{J} &= I - \mathcal{D}^{-1}\mathcal{A}, \\ &= I - (\text{diag}(I, P_2D_2P_2^T, \dots, P_kD_kP_k^T))^{-1}(PAP^T), \\ &= I - (\text{diag}(I, P_2D_2^{-1}P_2^T, \dots, P_kD_k^{-1}P_k^T))(PAP^T), \\ &= I - P(\text{diag}(I, D_2^{-1}, \dots, D_k^{-1}))P^T(PAP^T), \\ &= I - PD^{-1}AP^T = P(I - D^{-1}A)P^T = PJP^T. \end{aligned}$$ □

In view of the previous theorem and because J is block 2-cyclic consistently ordered, the EJ and the SOR theory holds, and the optimal values of the various parameters of the block EJ and the block SOR methods associated with $\mathcal{A}$ are those obtained for the matrix A in Theorems 3.2–3.5.

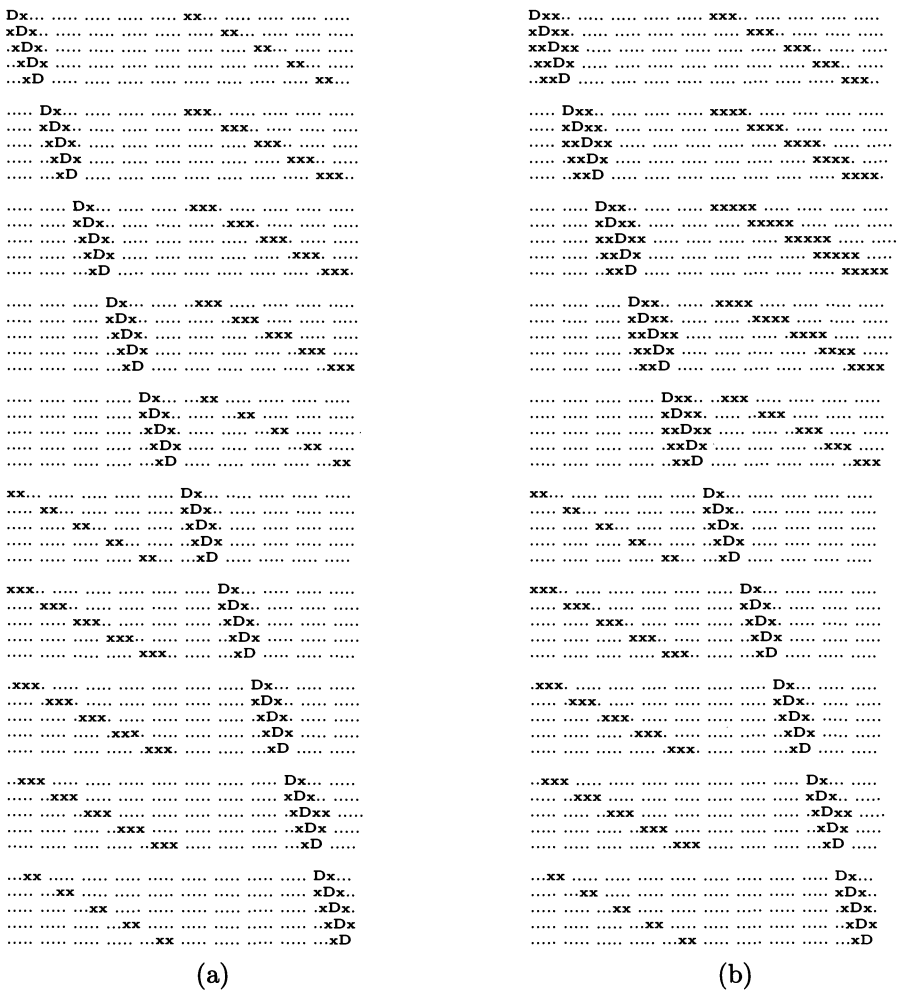


FIG. 4. *Structure of matrices from the* (a) *second-order and* (b) *fourth-order* LCSC *collocation methods for a two-dimensional problem. We have* $N_1 = N_2 = 6$ *with the notation* D = *diagonal nonzero element,* x = *off-diagonal nonzero element, and* . = *zero element.*

4. Numerical verification of convergence. In this section we present the results of some numerical experiments on the convergence properties of the LCSC method and the iterative solvers. These experiments verify the convergence properties of the LCSC discretization and the iterative solution methods of §3. They also indicate that these methods work equally well for more general problems than (1b), (2b) where our analysis does not apply. Specifically, we have applied the LCSC methods to approximate the known solution of the following PDE problems:

PDE A2: $D_x^2u + D_y^2u = f,$
PDE B2: $D_x^2u + 4D_y^2u - 10u = f,$
PDE C2: $D_x^2u + D_y^2u + 10e^{x+y}u = f,$
PDE A3: $D_x^2u + D_y^2u + D_z^2u = f,$
PDE B3: $D_x^2u + D_y^2u + 4D_z^2u - 10u = f,$
PDE C3: $D_x^2u + (x+z)D_y^2u + (4-x-z)D_z^2u + e^{x+y+z}u = f.$

All these problems are on the unit square or cube with homogeneous Dirichlet boundary conditions ($u = 0$), and f is such that the true solution is

$$u(x, y) = e^{x+y}(x^2 - x)(y^2 - y)$$

or

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

The approximate solutions of the discretized problem are determined by the iterative methods presented and analyzed in §3. For the LCSC discretization, we estimate the maximum error ($||u - u_h||_\infty$) at the mesh points and compute an estimate of its order of convergence from the expression

$$(50) \qquad -\log\left(\frac{||(u - u_{h_1})||_\infty}{||(u - u_{h_2})||_\infty}\right) \Big/ \log\left(\frac{h_1}{h_2}\right),$$

where h_1, h_2 are the grid steps for the two different uniform partitions. All the computations were done on a SUN 4/110 in double-precision arithmetic. The linear systems from the LCSC discretization are solved by the iteration of §3, with termination criterion being that $||\mathbf{U}^{(s+1)} - \mathbf{U}^{(s)}||_\infty$ is within the interval $(0, 10^{-7})$. Tables 1 and 3 show the results for the six problems and the second-order LCSC method using an (NGRID by NGRID) or (NGRID by NGRID by NGRID) mesh for NGRID = 8, 16, 24, ... We see that the discretization error decreases regularly, and that the estimated order of convergence agrees well with the asymptotic result expected from Theorem 2.1. Similarly, Tables 2 and 4 show the results for the fourth-order LCSC discretization and, again, the estimated order of convergence agrees well with that expected from Theorem 2.1.

Tables 5 and 6 present the number of SOR iterations required to solve the discretized equations using a 10^{-7} stopping criterion for the second- and fourth-order LCSC methods. We see that the number of iterations increases a little faster than linearly in NGRID.

The number of iterations for the fourth-order case increases more rapidly as NGRID increases, apparently faster than linearly, but slower than quadratically. Of course, the fourth-order discretization is much more efficient as it achieves much higher accuracy. Restated, for PDE B2, we could use the fourth-order discretization with an 8×8 mesh and 132 iterations instead of the second-order discretization with a 24×24 mesh and 279 iterations. Both ways, we achieve an accuracy of about .006. It is important to note that the number of iterations does *not* increase going from two to three dimensions.

We also note in all these tables that the analytical results established for the Helmholtz problem are observed for the non-Helmholtz problems PDE C2 and PDE C3.

The nature of the convergence of the iterative methods is examined in more detail in Figures 5–7.

Figures 5–7 explore the accuracy of the theoretical predictions of the optimum relaxation factor ω_{opt} for three cases: second-order LCSC discretization in two dimensions (Fig. 5), fourth-order LCSC discretization in two dimensions (Fig. 6), and

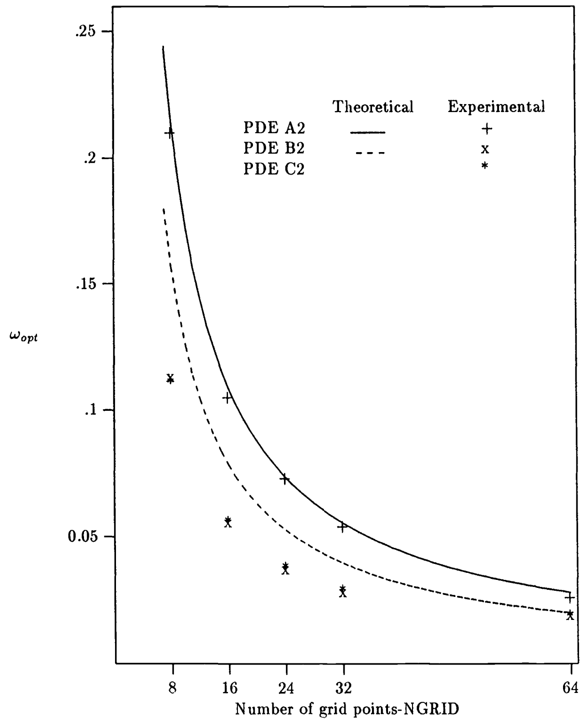


FIG. 5. *Graph of the ω_{opt} vs. the number of grid points for the second-order* LCSC *method applied to* PDE A2, PDE B2, *and* PDE C2.

second-order LCSC discretization in three dimensions (Fig. 7). The experimental values of ω_{opt} are determined by systematic search. In all cases we see that the theoretical optimum values are good estimates of the actual optimums. Furthermore, the theoretical ω_{opt} values for the Helmholtz problems are good estimates for the more general operator problems where our theory does not apply.

TABLE 1

The error and estimated order of convergence of the second-order LCSC *method for the three elliptic problems* PDE A2, PDE B2, *and* PDE C2. *The convergence* (order) *of the discretization is estimated by* (50) *from successive values of* $h = 1/\text{NGRID}$.

	PDE A2		PDE B2		PDE C2	
NGRID	error	order	error	order	error	order
8	.86 D−2		.73 D−2		.22 D−1	
16	.22 D−2	1.79	.18 D−2	1.83	.57 D−2	1.95
24	.96 D−3	1.94	.73 D−3	2.11	.25 D−2	2.03
32	.55 D−3	1.87	.38 D−3	2.13	.13 D−3	2.17
64	.14 D−3	1.93	.10 D−3	1.89	.30 D−3	2.06

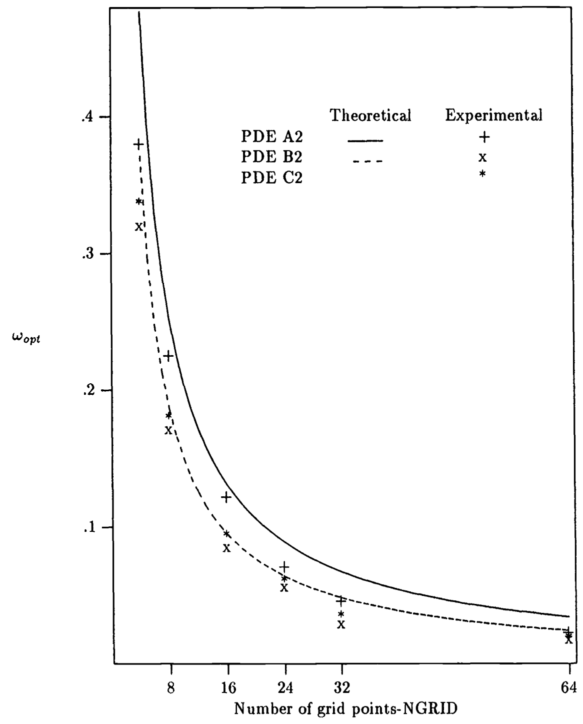


FIG. 6. *Graph of the* ω_{opt} *vs. the number* NGRID *of grid points for the fourth-order* LCSC *method applied to* PDE A2, PDE B2, *and* PDE C2.

TABLE 2

The error and estimated order of convergence of the fourth-order LCSC *method for the three elliptic problems* PDE A2, PDE B2, *and* PDE C2. *The convergence* (order) *of the discretization is estimated by* (50) *from successive values of* $h = 1/\text{NGRID}$.

NGRID	PDE A2 error	PDE A2 order	PDE B2 error	PDE B2 order	PDE C2 error	PDE C2 order
8	.69 D−3		.54 D−3		.92 D−3	
16	.54 D−4	3.67	.48 D−4	3.49	.64 D−4	3.85
24	.12 D−4	3.71	.97 D−5	3.94	.14 D−4	3.74
32	.39 D−5	3.91	.30 D−5	4.07	.42 D−5	4.12
64	.25 D−6	3.96	.22 D−6	3.77	.30 D−6	3.81

TABLE 3

The error and estimated order of convergence of the second-order LCSC *methods for three elliptic problems* PDE A3, PDE B3, *and* PDE C3. *The convergence* (order) *of the discretization is estimated by* (50) *using successive values of* $h = 1/\text{NGRID}$.

NGRID	PDE A3 error	PDE A3 order	PDE B3 error	PDE B3 order	PDE C3 error	PDE C3 order
8	.38 D−2		.33 D−2		.42 D−2	
16	.97 D−3	1.79	.87 D−3	1.75	.11 D−2	1.75
24	.56 D−3	1.90	.41 D−3	1.76	.72 D−3	1.82
32	.26 D−3	1.90	.24 D−3	1.81	.41 D−3	1.82

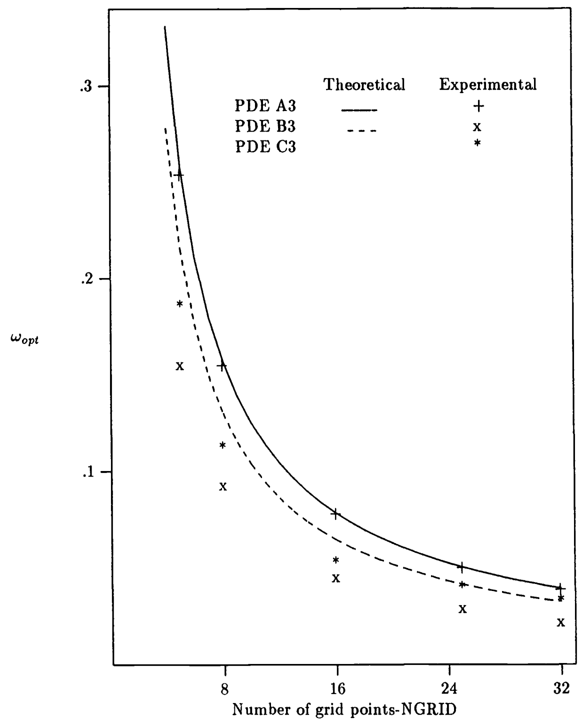


FIG. 7. *Graph of the* ω_{opt} *vs. the number* NGRID *of grid points for the second-order* LCSC *method applied to the three-dimensional problems* PDE A3, PDE B3, *and* PDE C3.

TABLE 4

The error and estimated order of convergence of the fourth-order LCSC *methods for the three elliptic problems* PDE A3, PDE B3 *and* PDE C3. *The convergence* (order) *of the discretization is estimated by* (50) *using successive values of* $h = 1/\text{NGRID}$.

NGRID	PDE A3 error	PDE A3 order	PDE B3 error	PDE B3 order	PDE C3 error	PDE C3 order
8	.49 D−3		.41 D−3		.76 D−3	
16	.51 D−4	3.28	.40 D−4	3.35	.86 D−4	3.14
24	.98 D−5	4.01	.85 D−5	3.82	.20 D−4	3.61
32	.33 D−5	3.79	.30 D−5	3.65	.67 D−5	3.82

TABLE 5

The required number of iterations of the SOR *method to solve the second- and fourth-order* LCSC *equations for the elliptic problems* PDE A2, PDE B2, *and* PDE C2 *using* 10^{-7} *as iteration termination criterion.*

	Second-order LCSC			Fourth-order LCSC		
NGRID	PDE A2	PDE B2	PDE C2	PDE A2	PDE B2	PDE C2
8	48	101	97	61	132	105
16	97	201	188	111	228	210
24	137	279	268	196	345	293
32	185	367	353	389	520	478

TABLE 6
The required number of iterations of the SOR *method to solve the second- and fourth-order* LCSC *equations for the elliptic problems* PDE A3, PDE B3, *and* PDE C3 *using* 10^{-7} *as iteration termination criterion.*

	Second-order LCSC			Fourth-order LCSC		
NGRID	PDE A3	PDE B3	PDE C3	PDE A3	PDE B3	PDE C3
8	60	114	92	82	135	110
16	122	219	193	150	273	225
24	170	330	327	242	417	310
32	237	442	402	393	710	512

Acknowledgment. We would like to thank Bob Lynch for his constructive remarks.

REFERENCES

[1] G. AVDELAS AND A. HADJIDIMOS, *Optimum accelerated overrelaxation method in a special case*, Math. Comp., 36 (1981), pp. 183–187.
[2] J. DE PILLIS AND M. NEUMANN, *Iterative methods with k-part splittings*, IMA J. Numer. Anal., 1 (1981), pp. 65–79.
[3] A. HADJIDIMOS, *The optimal solution of the extrapolation problem of a first order scheme*, Internat. J. Comput. Math., 13 (1983), pp. 153–168.
[4] ———, *On the extrapolation technique for the solution of linear systems*, Calcolo, XXIII (1986), pp. 35–43.
[5] ———, *A survey of the iterative methods for the solution of linear systems by extrapolation, relaxation and other techniques*, J. Comp. Appl. Math., 20 (1987), pp. 213–230.
[6] P. R. HALMOS, *Finite Dimensional Vector Spaces*, Van Nostrand, New York, 1958.
[7] E. N. HOUSTIS, J. R. RICE, AND E. A. VAVALIS, *Spline collocation methods for elliptic partial differential equations*, in Advances in Computer Methods for Partial Differential Equations, Vol. V, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, NJ, 1984, pp. 191–194.
[8] E. N. HOUSTIS, E. A. VAVALIS, AND J. R. RICE, *Convergence of* $O(h^4)$ *cubic spline collocation methods for elliptic partial differential equations*, SIAM J. Numer. Anal., 6 (1988), pp. 54–74.
[9] B. KREDELL, *On complex successive overrelaxation*, BIT, 2 (1962), pp. 143–152.
[10] R. E. LYNCH, J. R. RICE, AND D. H. THOMAS, *Tensor product analysis of partial differential equations*, Bull. Amer. Math. Soc., 70 (1964), pp. 378–384.
[11] N. M. MISSIRLIS, *Convergence theory of extrapolated iterative methods for a certain class of nonsymmetric linear systems*, Numer. Math., 45 (1984), pp. 447–458.
[12] W. NIETHAMMER, *Uberrelation bei Linearen Gleichungsystem mit Schiefsymetricher Koeffizientenmatrix*, Ph.D. thesis, University of Tubingen, West Germany, 1964.
[13] G. D. SMITH, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, 2nd ed., Calerdon Press, London, 1978.
[14] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1962.
[15] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 735–751, May 1993

AN ALGORITHM FOR SYMMETRIC TRIDIAGONAL EIGENPROBLEMS: DIVIDE AND CONQUER WITH HOMOTOPY CONTINUATION[*]

KUIYUAN LI[†] AND TIEN-YIEN LI[‡]

Abstract. This paper presents a new algorithm for finding all the eigenvalues and corresponding eigenvectors of a symmetric tridiagonal matrix. The algorithm is based on the homotopy continuation approach coupled with the strategy of "divide and conquer." Evidenced by the numerical results, the algorithm given here provides a considerable advance over previous attempts to use the homotopy method for eigenvalue problems. Numerical comparisons of this algorithm with the methods in the widely used EISPACK library, as well as Cuppen's divide and conquer method, are presented. It appears that the algorithm is strongly competitive in terms of speed, accuracy, and orthogonality. The performance of the parallel version of this algorithm is also presented. The natural parallelism of the algorithm makes it an excellent candidate for a variety of advanced architectures.

Key words. eigenvalues, eigencurves, multiprocessors, homotopy method

AMS(MOS) subject classification. 65F15

1. Introduction.

In this paper, we propose a new algorithm, based on the continuation approach, for finding all the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. Let A be an $n \times n$ real symmetric tridiagonal matrix of the form

$$A = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}. \tag{1.1}$$

In (1.1), if some $\beta_i = 0$, then $\mathbf{R^n}$ is clearly decomposed into two complementary subspaces invariant under A. Thus the eigenproblem is decomposed in an obvious way into two smaller subproblems. Therefore, we assume that each $\beta_i \neq 0$. That is, A is *unreduced.* Our algorithm uses the "divide and conquer" strategy. First, the matrix A is divided into two blocks by letting one of the β_i's equal to zero. Namely, we let

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}, \tag{1.2}$$

* Received by the editors December 10, 1990; accepted for publication (in revised form) April 2, 1992.

† Department of Mathematics and Statistics, University of West Florida, Pensacola, Florida 32514.

‡ Department of Mathematics, Michigan State University, East Lansing, Michigan 48824. The research of this author was supported in part by National Science Foundation grant CCR-9024840.

where

$$D_1 = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{pmatrix}, \quad D_2 = \begin{pmatrix} \alpha_{k+1} & \beta_{k+2} & & & \\ \beta_{k+2} & \alpha_{k+2} & \beta_{k+3} & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}.$$

We then calculate the eigenvalues of unreduced matrices D_1 and D_2 by using the most efficient algorithm available. Unlike Cuppen's divide and conquer method [4], our algorithm conquers the matrix A by the homotopy $H : \mathbf{R^n} \times \mathbf{R} \times [\mathbf{0}, \mathbf{1}] \longrightarrow \mathbf{R^n} \times \mathbf{R}$, defined by

$$\begin{aligned} H(x, \lambda, t) &= (1-t)\begin{pmatrix} \lambda x - Dx \\ \dfrac{x^T x - 1}{2} \end{pmatrix} + t\begin{pmatrix} \lambda x - Ax \\ \dfrac{x^T x - 1}{2} \end{pmatrix} \\ &= \begin{pmatrix} \lambda x - [(1-t)D + tA]x \\ \dfrac{x^T x - 1}{2} \end{pmatrix} \\ &= \begin{pmatrix} \lambda x - A(t)x \\ \dfrac{x^T x - 1}{2} \end{pmatrix}, \end{aligned} \tag{1.3}$$

where $A(t) = (1-t)D + tA$. It can easily be seen that the solution set of $H(x, \lambda, t) = 0$ in (1.3) consists of disjoint smooth curves $(x(t),\ \lambda(t))$, each joining an eigenpair of D to one of A. We call each of these curves a *homotopy curve* or an *eigenpath* and its component $\lambda(t)$ an *eigenvalue path.* Thus, by following the eigenpaths emanating from the eigenpairs of D at $t = 0$, we can reach all the eigenpairs of A at $t = 1$.

Unlike curve-following strategies used in standard homotopy continuation methods, our algorithm mainly uses the homotopy continuation as a *backup* of Newton's method. Simple computation shows that Newton's method for the nonlinear problem of $n + 1$ equations

$$F(\lambda, x) = \begin{cases} \lambda x - Ax = 0, \\ \dfrac{x^T x - 1}{2} = 0 \end{cases} \tag{1.4}$$

in the $n + 1$ variables λ, $x_1, x_2, \ldots, x_n$ at $(\lambda^{(m)}, x^{(m)})$ is the inverse iteration of A on $x^{(m)}$ with shift $\lambda^{(m)}$, i.e.,

$$(A - \lambda^{(m)} I)y = x^{(m)}$$

and

$$\begin{cases} \lambda^{(m+1)} = \lambda^{(m)} + \dfrac{(x^{(m)})^T x^{(m)} + 1}{2(x^{(m)})^T y}, \\ x^{(m+1)} = (\lambda^{(m+1)} - \lambda^{(m)})y. \end{cases}$$

By making the initial matrix D close to A, the eigenpairs of D should be excellent starting points for applying Newton's method to the eigenproblem (1.4). Based on this observation, our algorithm solves the eigenvalues of the initial matrix D by using the most efficient method available first, and applying the inverse iteration with a random starting vector on (1.4), using each eigenvalue of D as a shift. This is essentially the first step of the usual curve-following scheme to follow the eigenpath $(x(t), \lambda(t))$ of the homotopy $H(x, \lambda, t) = 0$ in (1.3) with starting stepsize $h = 1$. After the first inverse iteration, we switch to Rayleigh quotient iteration (RQI), an inverse iteration with Rayleigh quotient as a shift, to speed up the convergence. We check the Sturm sequence at the convergent point and, if this procedure fails to provide the correct eigenpair, we cut the stepsize in half. That is, we repeat the process by applying the inverse iteration to (1.3) with $t = 0.5$ and then switch to RQI to return to the right eigenpath $(x(t), \lambda(t))$. Assuming that after i steps, the approximate value $(x(t_i), \lambda(t_i))$ is known, we always choose stepsize $h = 1 - t_i$ at $(x(t_i), \lambda(t_i))$. In this way, we follow the eigenpath from $t = 0$ to $t = 1$. A theoretical analysis of the eigenpath is given in §2, and the details of the algorithm are described in §3.

The search for fast, reliable methods for handling symmetric eigenproblems has produced a number of methods, most notably the QR algorithm, the bisection Sturm sequence method with inverse iteration [6], [7] and the divide and conquer method [4], [5], [15]. In §4.1 we present our numerical results, along with comparisons to results obtained with those methods. It appears that our algorithm is strongly competitive in terms of speed, accuracy, and orthogonality, and leads in speed in almost all cases.

Modern scientific computing is marked by the advent of vector and parallel computers and the search for algorithms that are, to a large extent, parallel in nature. A further advantage of our method is that, to a large degree, it is parallel in the sense that each eigenpath is followed independently of the others. It makes the parallel implementation of the homotopy method much simpler than the other methods. In §4.2, we show the performance of our parallel algorithm. The results suggest that our algorithm may be an excellent candidate for a variety of architectures.

The theoretical aspect of the continuation approach to the eigenvalue problems has been studied in [1]–[3] and [9]. A first attempt was made in [8] to make the method computationally efficient. Its parallel version appeared in [11]. Evidenced by the numerical results, our algorithms given here provide a considerable improvement over the algorithms in [8] and [11].

2. Preliminary analysis.

PROPOSITION 2.1. *Let $\lambda(t)$ be an eigenvalue path of the homotopy $H(x, \lambda, t) = 0$ in (1.3). Then, either $\lambda(t)$ is constant for all $t \in [0, 1]$ or strictly monotonic. (See Fig. 1.)*

Proof. For a symmetric tridiagonal matrix

$$M = \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & b_n \\ & & & b_n & a_n \end{pmatrix},$$

with $b_i \neq 0$, i =2, 3,..., n, let M_i be the $i \times i$ principle submatrix of M. Let

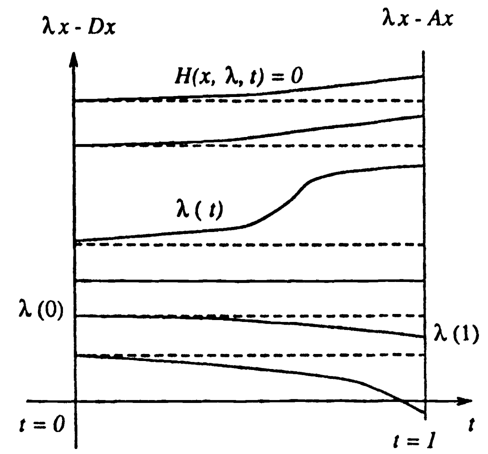


FIG. 1

$p_i(t) = \det(M_i - \lambda I)$. The well-known three-terms recurrence relation gives [13]

$$p_{i+1}(\lambda) = (a_{i+1} - \lambda)p_i(\lambda) - b_{i+1}^2 p_{i-1}(\lambda). \tag{2.1}$$

Applying (2.1) to

$$A(t) = (1-t)D + tA = \left(\begin{array}{cc|cc} D_1 & & & \\ & & t\beta_{k+1} & \\ \hline & t\beta_{k+1} & & \\ & & & D_2 \end{array}\right)$$

in our homotopy in (1.3), we can easily see that the variable t appears in $\det(A(t) - \lambda I)$ only in the second degree. Namely,

$$p(\lambda, t) \equiv \det(A(t) - \lambda I) = t^2 f_1(\lambda) + f_2(\lambda)$$

for certain polynomials $f_1(\lambda)$ and $f_2(\lambda)$. Thus, for any given λ_0, if $f_1(\lambda_0) \neq 0$, then $p(\lambda_0, t) = 0$ has two solutions

$$t = \pm\sqrt{-\frac{f_2(\lambda_0)}{f_1(\lambda_0)}}.$$

Thus, when $-f_2(\lambda(t_0))/f_2(\lambda(t_0)) > 0$, t is real and $p(\lambda_0, t) = 0$ can have at most one solution in [0,1]. It follows that the eigenvalue path $\lambda(t)$ of (1.3), a solution of $p(\lambda, t) = 0$ for each t in [0,1], is strictly monotonic. Otherwise, for certain values of λ_0, $p(\lambda_0, t) = 0$ will have more than one solution in [0,1]. (See Fig. 2.)

If an eigenvalue λ_0 of the initial matrix D satisfies $f_1(\lambda_0) = 0$, then $p(\lambda_0, 0) = \det(D - \lambda_0 I) = 0$ implies $f_2(\lambda_0) = 0$. Hence, $p(\lambda_0, t) = 0$ for all t in [0,1] and $\lambda(t) \equiv \lambda_0$ gives a constant eigenvalue path.

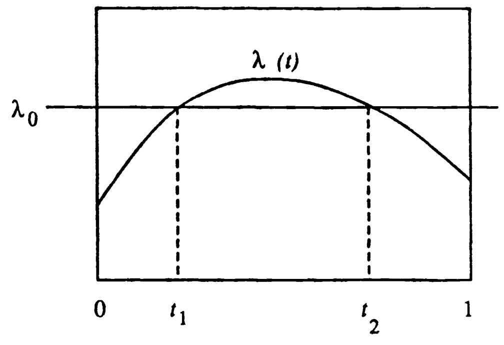


FIG. 2

Remark. It follows from the proof of the above proposition that if

$$\lambda_1(0) \leq \lambda_2(0) \leq \cdots \leq \lambda_n(0),$$

then

$$\lambda_{i-1}(0) \leq \lambda_i(t) \leq \lambda_{i+1}(t), \qquad 2 \leq i \leq n-1.$$

THEOREM 2.2 (HOFFMAN AND WIELANDT [16]). *Let M be an $n \times n$ symmetric matrix. Let $M' \equiv M + E$ where E is a symmetric perturbation of M. Denote the eigenvalues of M by $\{\xi_1 \leq \xi_2 \leq \cdots \leq \xi_n\}$, the eigenvalues of M' by $\{\xi_1' \leq \xi_2' \leq \cdots \leq \xi_n'\}$, and the eigenvalues of E by $\{\gamma_1 \leq \gamma_2 \leq \cdots \leq \gamma_n\}$, then*

$$\sum_{i=1}^{n} (\xi_i - \xi_i')^2 \leq \sum_{i=1}^{n} \gamma_i^2. \tag{2.2}$$

Applying this theorem to $M = A$ and $M' = A(t) = (1-t)D + tA = A + E$ with

$$E = \left(\begin{array}{c|c} O & (t-1)\beta_{k+1} \\ \hline (t-1)\beta_{k+1} & O \end{array} \right),$$

then (2.2) gives

$$\sum_{i=1}^{n} (\lambda_i - \lambda_i(t))^2 \leq 2(1-t)^2 \beta_{k+1}^2 \qquad \text{for all } t \text{ in } [0,1], \tag{2.3}$$

where λ_i and $\lambda_i(t)$ are the eigenvalues of A and $A(t)$, respectively.

From (2.3), together with the conclusion in Proposition 2.1 that each nonconstant $\lambda_i(t)$ is monotonic in t, we see that the smaller β_{k+1} is, the *flatter* the eigenvalue curves are, especially when n is very large. To make the eigencurves easy to follow, we intend to choose β_{k+1} as small as we can for k in a certain range, as described in §3.

3. Algorithms. The basic features of our algorithm to follow the eigenpath $(x(t), \lambda(t))$ are

(i) Initiating at $t = 0$,
(ii) Prediction,
(iii) Correction,
(iv) Checking,
(v) Detection of a cluster and space iteration,
(vi) Stepsize selection,
(vii) Terminating at $t = 1$.

In this section, we give a detailed description of these features.

(i) *Initiating at* $t = 0$. As mentioned in §2, we intend to choose k for which β_{k+1} is as small as possible. To make the sizes of the blocks D_1 and D_2 roughly the same, we limit the choice of k in the range $n/2 - \delta \le k \le n/2 + \delta$, where $\delta \approx n/20$ and find the smallest β_{k+1} by local sorting.

When the initial matrix D is decided, different from the homotopy algorithms in [8], [11] where all the eigenvalues and eigenvectors of D are calculated in order to start following the eigenpaths, our algorithm only calculates the eigenvalues of D_1 and D_2. These eigenvalues are obtained by using the most efficient method available. From our experience, for matrices of relatively small size, the QR algorithm is the fastest method for this purpose. Therefore, in the real implementation, we use the subroutine TQL1 in EISPACK [14] here. (Since the QL algorithm in TQL1 is highly serial, it is not very efficient with a modest number of processors on large problems. Thus, for parallel computing, if a modest number of processors are available, we use the multisection method. Otherwise, we still use TQL1.) At this stage, the precision of the eigenvalues of D_1 and D_2 is not crucial. Thus we only require that the accuracy stay within one-half or even one-third of the working precision. With this strategy, a considerable amount of computing time is reduced.

(ii) *Prediction.* Assume that after i steps the approximate value $(\tilde{x}(t_i), \tilde{\lambda}(t_i))$ on the eigenpath $(x(t), \lambda(t))$ at t_i is known and the next stepsize h is determined; that is, $t_{i+1} = t_i + h$. We want to find an approximate value $(\tilde{x}(t_{i+1}), \tilde{\lambda}(t_{i+1}))$ of $(x(t_{i+1}), \lambda(t_{i+1}))$ on the eigenpath at t_{i+1}. Note that $(\tilde{x}(t_{i+1}), \tilde{\lambda}(t_{i+1}))$ is an approximate eigenpair of $A(t_{i+1})$. Since $H(x(t), \lambda(t), t) = 0$, we have

$$A(t)x(t) = \lambda(t)x(t), \quad x^T(t)x(t) = 1.$$

Differentiating both equations with respect to t yields,

$$\begin{aligned} (A - D)x(t) + A(t)\dot{x}(t) &= \dot{\lambda}(t)x(t) + \lambda(t)\dot{x}(t), \\ x^T(t)\dot{x}(t) &= 0. \end{aligned} \tag{3.1}$$

For $t = t_i$, multiplying (3.1) on the left by $x^T(t_i)$ yields,

$$\dot{\lambda}(t_i) = x^T(t_i)(A - D)x(t_i) = 2\beta_{k+1}x_k(t_i)x_{k+1}(t_i), \tag{3.2}$$

where $x(t_i) = (x_1(t_i), \ldots, x_n(t_i))^T$. With $\dot{\lambda}(t_i)$ at hand, we use the Euler predictor to predict the eigenvalue at t_{i+1}, namely,

$$\lambda^0(t_{i+1}) = \lambda(t_i) + \dot{\lambda}(t_i)h.$$

It is easy to see that $\dot{\lambda}(0) = 0$ in (3.2), since $x(0)$ is either of form $(x_1(0), \ldots, x_k(0), 0, \ldots, 0)$ or $(0, \ldots, 0, x_{k+1}(0), \ldots, x_n(0))$. Consequently, $\lambda^0(t_1)$ always equals $\lambda(0)$. To predict the eigenvector, we use the inverse power method of $A(t_{i+1})$ on $x(t_i)$ with shift $\lambda^0(t_{i+1})$. That is, we first solve

$$(3.3) \qquad (A(t_{i+1}) - \lambda^0(t_{i+1})I)y^0(t_{i+1}) = x(t_i),$$

and let

$$x^0(t_{i+1}) = \frac{y^0(t_{i+1})}{||y^0(t_{i+1})||}.$$

At $t_i = 0$, since we skip the calculations of eigenvectors of D, $x(0)$ is not available. We chose a random vector to substitute for $x(0)$ in (3.3).

(iii) *Correction.* As a corrector, we use the standard RQI, starting with $x^0(t_{i+1})$. To be more precise, at $(x^{j-1}(t_{i+1}), \lambda^{j-1}(t_{i+1}))(j \geq 1)$ let

$$\lambda^j(t_{i+1}) = x^{j-1}(t_{i+1})^T A(t_{i+1}) x^{j-1}(t_{i+1}),$$

then solve

$$(A(t_{i+1}) - \lambda^j(t_{i+1})I)y^j(t_{i+1}) = x^{j-1}(t_{i+1}),$$

and let

$$x^j(t_{i+1}) = \frac{y^j(t_{i+1})}{||y^j(t_{i+1})||}.$$

We repeat the above process to within half of the working precision if single precision is used, and one-third of the working precision if double precision is used when $t_{i+1} < 1$, since precision in determining the curve is only of secondary interest. We polish $(x^j(t_{i+1}), \lambda^j(t_{i+1}))$ at the end of the path $(t_{i+1} = 1)$ by iterating the Rayleigh quotient to machine precision. The stop point $(x^j(t_{i+1}), \lambda^j(t_{i+1}))$ of RQI will be taken as an approximate eigenpair $(\tilde{x}(t_{i+1}), \tilde{\lambda}(t_{i+1}))$ of $A(t_{i+1})$. The cubic convergence rate of RQI makes the corrector highly efficient.

(iv) *Checking.* When $(\tilde{x}(t_{i+1}), \tilde{\lambda}(t_{i+1}))$ is taken as an approximate eigenpair of $A(t_{i+1})$, the Sturm sequences at $\tilde{\lambda}(t_{i+1}) \pm \epsilon\tilde{\lambda}(t_{i+1})$ are computed to check that, if we are trying to follow the curve of jth highest eigenvalues, we are still on that curve. Here, ϵ is chosen as half of the working precision if single precision is used, and one-third of the working precision if double precision is used. If the checking fails, we reduce the stepsize to $h/2$ and repeat the whole process once again, beginning with the eigenvalue prediction in (ii).

(v) *Detection of a cluster and subspace iteration.* At $t_i = 0$, when all the eigenvalues of D, $\lambda_1(0) \leq \lambda_2(0) \leq \cdots \leq \lambda_n(0)$, are available, we let $\delta = \max(10^{-5}, 10^{-2}(\lambda_n(0) - \lambda_1(0))/n)$ if double precision is used (or $\delta = \max(10^{-3}, 10^{-2}(\lambda_n(0) - \lambda_1(0))/n)$ if single precision is used). Set λ_i and λ_j in the same group if $|\lambda_i(0) - \lambda_j(0)| < \delta$. If the number of the eigenvalues in any group is bigger than 1, then a cluster is detected. At $t_i \neq 0$, or 1, when $(\tilde{x}(t_i), \tilde{\lambda}(t_i))$ is taken as an approximate eigenpair of $A(t_i)$, after the checking step in (iv) we compute the Sturm sequences at $\tilde{\lambda}(t_i) \pm \delta$ for the purpose of finding the number of eigenvalues of $A(t_i)$ in the interval $(\tilde{\lambda}(t_i) - \delta, \tilde{\lambda}(t_i) + \delta)$. When this number is bigger than 1, a cluster of eigenvalues of $A(t_i)$ is detected.

In those cases, the corresponding eigenvectors are ill conditioned and this ill-conditioning can cause the inefficiency of the algorithm. To remedy this problem, the inverse power method [13], [16] with $\tilde{\lambda}(t_i)$ as shifts is used to construct an approximation of the corresponding eigenspace S of dimension m (= the number of eigenvalues in the cluster) of $A(t_i)$. This approximate eigenspace S is used as an initial subspace of the subspace iterations at t_{i+1} when we approximate the eigenpairs of $A(t_{i+1})$.

(vi) *Stepsize selection.* In the first attempt, we always choose stepsize $h = 1-t_i$ at $t_i < 1$. If, after the prediction and correction steps, the checking step fails, we reduce the stepsize to $h/2$ as mentioned in (iv). This extremely liberal choice of stepsize can be justified because of the closeness of the matrix D to A as well as the effective checking algorithm. Indeed, since the initial matrix D is chosen to be so close to A, from our experience, the majority of the eigenpairs of A can be reached in one step, i.e., $h = 1$.

Very small stepsize can also cause inefficiency of the algorithm. Therefore, we impose a minimum γ on stepsize h. If $h < \gamma$, we simply give up following the eigenpath, and the corresponding eigenpair of A will be calculated at the end of the algorithm by the method of bisection with inverse iterations (see (vii)). We usually choose $\gamma \approx 0.25$.

(vii) *Terminating at* $t = 1$. At $t = 1$, when an approximate eigenvalue $\tilde{\lambda}(1)$ is reached, we compute the Sturm sequence at $\tilde{\lambda}(1) \pm \epsilon\tilde{\lambda}(1)$ with ϵ = machine precision to ensure the correct order. If the checking fails, we have jumped into a wrong eigenpath. More precisely, suppose that as we are following the ith eigenpair, the checking algorithm detects that we have reached the jth eigenpair instead. In this situation, we will save the jth eigenpair before the stepsize is cut. By saving the jth eigenpair, the computation of following the jth eigenpair is no longer needed.

As mentioned in (vi), we may give up following some eigenpaths to avoid adopting a stepsize that is too small. Without extra computation, we know exactly which eigenpairs are lost at $t = 1$. In order to find these eigenpairs, we first use the bisection to find the missing eigenvalues up to half the working precision and then use the inverse iteration and the RQI or subspace iteration (if there are some clusters) to find the eigenpairs.

4. Numerical results.

4.1. A serial comparison with the existing methods.

For symmetric tridiagonal matrices, the routine TQL2 in EISPACK implements explicit QL iteration to find all the eigenpairs. EISPACK also includes a Sturm sequence with inverse iteration method (BISECT+TINVIT), which is much faster than TQL2. However, it may fail to provide more accurate eigenvectors when the corresponding eigenvalues are very close. A new version by Jessup [7] (B/III) has considerably improved the reliability and the accuracy of the inverse iteration. The divide and conquer method for symmetric tridiagonal matrix was suggested by Cuppen [4] and was implemented, combined with a deflation and a robust root-finding technique, by Dongarra and Sorensen [5] (TREEQL) (see also [15]).

The homotopy continuation algorithm is in its preliminary stage, and much development and testing are necessary. But the numerical results on the examples we have looked at seem remarkable. The standard testing matrices are:

(1) The Toeplitz matrix [1,2,1].

(2) The random matrix with both diagonal and off-diagonal elements being uniformly distributed random numbers between 0 and 1.

(3) The Wilkinson matrix W_n^+, i.e., the matrix $[1, d_i, 1]$, where $d_i = \text{abs}((n + 1)/2 - i)$, $i = 1, 2, \ldots, n$ with n odd.

(4) The matrix $[1,\mu_i,1]$, where $\mu_i = i \times 10^{-6}$.

(5) The matrix T_2: same as matrix [2,8,2] except the first diagonal element $\alpha_1 = 4$.

(6) BW matrix: the matrix consists of two copies of Wilkinson matrices W_k^+ with even k along the diagonal and the off-diagonal elements at the position β_{k+1}, where the submatrices join equal to 10^{-6}.

We show the computational results comparing our homotopy continuation algorithm HOMO to those methods, TQL2, B/III, and TREEQL, mentioned above. The computations were done on a Sun SPARC station 1.

Tables 1, 2, and 3 show the comparisons in terms of speed, accuracy, and orthogonality, respectively. The homotopy method appears to be strongly competitive in every category and leads in speed by a considerable margin in comparison with all other methods in most of the cases.

Table 4 gives an analysis of our algorithm on those testing matrices. The fourth column on the table shows the total number of stepsize cuttings because of the curve jumping. The fifth column shows how many clusters occurred and the last column shows the largest dimension of the subspaces of the clusters.

As we mentioned in §§2 and 3, to form the initial matrix D, we usually choose the smallest off-diagonal entry β_{k+1}, for k in a certain range, of A, making it zero in D so that D is as close to A as possible. This strategy is crucial for the efficiency of our algorithm. To illustrate this, we perform our algorithm on Toeplitz matrices $[\beta, i, \beta]$. Table 5 shows that our algorithm is much faster when β is smaller.

TABLE 1
Execution time (seconds) of computed eigenvalues and eigenvectors.

Matrix	Order	Execution time (seconds)			
	n	HOMO	B/III	TREEQL	TQL2
$[1,2,1]$	64	2.03	2.43	5.90	17.06
	125	5.88	8.58	37.61	114.4
	256	30.25	35.7	302.8	904.6
	499	100.04	152.9	984.4	2416.
Random	64	1.13	2.44	6.09	17.32
	125	3.79	8.53	31.41	115.8
	256	14.90	34.45	158.1	949.9
	499	53.19	133.6	235.8	2482.
W_n^+	65	0.97	1.84	2.73	16.10
	125	3.97	6.21	8.20	108.8
	255	16.40	22.57	31.83	879.9
	499	57.35	95.50	57.83	3869.
$[1,\mu_i,1]$	64	1.97	2.43	5.91	17.15
	125	6.78	8.64	37.65	115.6
	256	30.61	34.19	303.4	901.8
	499	107.04	129.3	984.4	2424.
T_2	64	1.80	2.39	5.80	16.68
	125	6.85	8.60	37.37	115.0
	256	28.24	34.86	165.2	939.8
	499	108.4	174.7	979.7	2506.
BW	64	1.80	1.73	2.07	12.20
	128	8.69	6.07	6.90	67.56
	256	38.85	22.51	29.07	490.7
	512	144.0	89.22	162.0	1216.

TABLE 2
The residual of computed eigenvalues and eigenvectors.

Matrix	Order	$\max_i \|\|Ax_i - \lambda_i x_i\|\|_2/\lambda_{\max}$			
	N	HOMO	B/III	TREEQL	TQL2
	64	1.91D−15	4.97D−16	1.90D−15	9.83D−15
$[1,2,1]$	125	1.95D−15	8.42D−16	3.36D−15	1.09D−14
	256	2.35D−15	1.28D−15	6.09D−15	2.75D−14
	499	2.22D−15	1.76D−15	7.51D−15	3.75D−14
	64	1.93D−16	3.66D−16	2.45D−14	6.05D−15
Random	125	2.04D−16	3.56D−16	5.65D−14	1.26D−14
	256	3.98D−15	3.62D−16	7.14D−14	5.39D−14
	499	1.03D−13	9.24D−15	4.82D−14	5.75D−14
	65	5.41D−16	5.49D−15	1.88D−13	8.79D−14
W_n^+	125	3.39D−16	7.16D−15	1.08D−12	3.44D−13
	255	7.78D−16	1.43D−14	1.08D−12	6.71D−13
	499	9.10D−16	2.83D−14	1.62D−11	6.57D−12
	64	4.91D−15	5.08D−16	2.44D−15	8.81D−15
$[1,\mu_i,1]$	125	3.22D−15	7.08D−16	3.81D−15	1.11D−14
	256	4.40D−15	1.14D−15	5.42D−15	2.75D−14
	499	4.94D−15	1.75D−15	8.44D−15	3.71D−14
	64	7.16D−16	1.77D−15	5.59D−15	1.58D−14
T_2	125	7.30D−16	1.85D−15	6.55D−15	2.67D−14
	256	8.17D−16	2.29D−15	1.34D−14	5.01D−14
	499	8.08D−16	3.73D−15	1.76D−14	8.17D−14
	64	8.15D−16	3.25D−15	4.38D−14	3.19D−14
BW	128	1.25D−15	1.30D−14	4.32D−13	1.14D−13
	256	2.13D−15	1.40D−14	3.84D−12	6.67D−13
	512	4.23D−15	2.38D−14	3.84D−12	1.76D−12

We also execute our algorithm on 21 types of testing matrices in newly established LAPACK, a package that rewrites the widely used LINPACK and EISPACK libraries to make them efficient on vector and parallel computers. Tables 6 and 7 show the comparisons of HOMO with B/III (DSTEBZ+DSTEIN in LAPACK). Types 1–7 of those testing matrices are diagonal matrices, and Table 6 shows that both algorithms work very well. The matrices of types 9, 17, and 21 have a large cluster with dimensions around $4n/5$ where n is the order of the matrices. For these matrices, HOMO is not as fast as B/III. The matrices of types 10 and 18 have an even larger cluster with dimension around $n-2$. Although HOMO still works, time consumption does not compare with B/III. We have a similar result for matrices of type

$$\begin{pmatrix} 0 & \mu & & & \\ \mu & 0 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & 0 & 1 \\ & & & 1 & 0 \end{pmatrix},$$

where μ is so large that HOMO detects a cluster of size $n-2$. When $n = 250$, while B/III takes about 150 seconds to find all the eigenpairs, HOMO needs about 1000 seconds. Tables 6 and 7 show that HOMO leads in speed for the other types of testing matrices.

TABLE 3
The orthonormality of computed eigenvectors.

Matrix	Order	$\max_{i,j} \lvert (X^T X - I)_{i,j} \rvert$			
	N	HOMO	B/III	TREEQL	TQL2
	64	6.21D−15	9.40D−15	1.48D−15	5.99D−15
[1, 2, 1]	125	5.06D−14	3.51D−14	3.62D−15	7.54D−15
	256	4.11D−14	2.10D−14	1.49D−14	1.59D−14
	499	1.89D−13	4.19D−14	1.95D−14	2.48D−14
	64	4.16D−15	2.66D−15	1.08D−14	7.32D−15
Random	125	1.75D−13	5.58D−15	4.15D−14	1.13D−14
	256	7.81D−13	3.99D−15	1.92D−13	3.64D−14
	499	1.94D−13	9.24D−15	4.82D−14	4.24D−14
	65	1.63D−15	2.22D−15	1.11D−15	7.54D−15
W_n^+	125	2.16D−15	3.21D−15	1.25D−15	1.26D−14
	255	1.36D−15	4.32D−15	4.44D−15	2.50D−14
	499	1.02D−15	1.22D−13	8.21D−15	6.97D−14
	64	4.28D−15	7.66D−15	1.08D−15	4.66D−15
$[1, \mu_i, 1]$	125	2.46D−14	4.33D−14	1.27D−14	6.66D−15
	256	7.63D−14	1.66D−13	8.40D−15	1.33D−14
	499	2.38D−13	1.39D−13	6.91D−14	2.44D−14
	64	1.14D−15	7.91D−15	9.35D−16	5.99D−15
T_2	125	3.64D−15	1.38D−14	3.74D−15	1.33D−14
	256	5.34D−14	1.99D−14	1.04D−14	2.17D−14
	499	1.01D−13	1.57D−14	1.91D−14	4.92D−14
	64	2.13D−15	3.55D−15	6.66D−16	7.99D−15
BW	128	1.36D−15	1.47D−13	1.13D−15	9.10D−15
	256	3.42D−15	1.03D−13	4.44D−15	2.37D−14
	512	1.95D−15	1.42D−13	1.13D−14	2.66D−14

4.2. Numerical results in the parallel case. Scientific and engineering research has become increasingly dependent upon the development and implementation of efficient parallel algorithms on modern high-performance computers. The search for algorithms for advanced computers suitable for eigenvalue problems has produced several algorithms, such as divide and conquer (D&C)[5] and bisection/multisection (B/M)[12] for symmetric tridiagonal matrices.

The homotopy algorithm is, to a large degree, parallel since each eigenpath can be followed independently. This inherent nature of the homotopy method makes the parallel implementation much simpler than other methods.

In our parallel algorithm, after all the eigenvalues of D are computed and put in increasing order, we assign each processor to trace roughly n/p eigencurves, where n is the order of matrix A and p is the number of the processors being used. Let the first processor trace the first n/p smallest eigencurves from the smallest to the largest, and let the second processor trace the second n/p smallest eigencurves, and so on.

We present, in this section, the numerical results of the parallel implementation of our algorithm. All examples were executed on the BUTTERFLY GP 1000, a shared-memory multiprocessor machine.

The *speedup* is defined as

$$S_p = \frac{\text{execution time using one processor}}{\text{execution time using } p \text{ processors}}$$

and the *efficiency* is the ratio of the speedup over p.

TABLE 4
Analysis of the homotopy algorithm.

Matrix	Order n	Execution time	Number of stepsize cutting	Number of clusters	Maximum dimension of subspaces of clusters
$[1,2,1]$	64	2.03	10	32	2
	125	5.88	28	4	2
	256	30.25	37	128	2
	499	100.04	111	22	2
Random	64	1.13	0	8	3
	125	3.79	4	8	4
	256	14.90	5	2	4
	499	53.19	1	6	5
W_n^+	65	0.97	1	25	2
	125	3.97	1	59	2
	255	16.40	1	121	2
	499	57.35	0	246	2
$[1,\mu_i,1]$	64	1.97	10	32	2
	125	6.78	28	5	2
	256	30.61	8	128	2
	499	107.04	113	10	2
T_2	64	1.80	11	2	2
	125	6.85	28	4	2
	256	28.24	38	11	2
	499	108.46	93	17	2
BW	64	1.80	0	20	4
	128	8.69	0	36	4
	256	38.85	2	68	4
	512	144.05	2	132	4

TABLE 5
Analysis of the homotopy algorithm on $[\beta,i,\beta]$ matrices with $n = 124$.

Matrix	Execution time of B/III	Execution time of HOMO	No. of stepsize cutting	No. of clusters	Max. subspace dimension	No. of subspace iteration	No. of RQI
$[1,i,1]$	8.86	2.01	0	0		0	256
$[10,i,10]$	8.78	3.83	7	2	2	10	505
$[10^2,i,10^2]$	9.06	5.34	18	0		0	667
$[10^3,i,10^3]$	8.67	6.89	32	2	2	11	810
$[10^4,i,10^4]$	8.95	9.01	57	0		0	961

Table 8 shows the execution time and the speedup S_p, as well as the efficiency S_p/p of our algorithm HOMO on matrices [1,2,1] and T_2. The speedup of our method over TQL2, *TQL2/HOMO*, on [1,2,1] is also listed. Similar results on matrices $[1,\mu_i,1]$ and BW are shown in Table 9, and on random matrices, in Table 10. It appears that our homotopy algorithm is very efficient.

The eigenvalues of the initial matrices of our algorithm on the examples in Tables 8, 9, and 10 were computed by TQL1. Since TQL1 is highly serial, it is not very efficient with modest numbers of processors on large problems. Table 11 shows the results of HOMO on random matrices by computing the eigenvalues of the initial matrices with multisection method. From Tables 10 and 11, we can see that if only a small number of nodes are available, using TQL1 to compute the eigenvalues of the initial matrices is better than using the multisection method.

TABLE 6
Execution time (seconds) of computed eigenvalues and eigenvectors from type 1 *to* 10.

Matrix	Order	Execution time		$\max_i \|\|Ax_i - \lambda_i x_i\|\|_2/\lambda_{\max}$		$\max_{i,j} \|(X^TX - I)_{i,j}\|/\lambda_{\max}$	
type	n	HOMO	B/III	HOMO	B/III	HOMO	B/III
Matrix	32	0.00	0.00	0.0	0.0	0.0	0.0
type	64	0.02	0.03	0.0	0.0	0.0	0.0
1	128	0.07	0.10	0.0	0.0	0.0	0.0
	256	0.32	0.33	0.0	0.0	0.0	0.0
	500	1.17	1.40	0.0	0.0	0.0	0.0
Matrix	32	0.01	0.01	0.0	0.0	0.0	0.0
type	64	0.02	0.02	0.0	0.0	0.0	0.0
2	128	0.08	0.09	0.0	0.0	0.0	0.0
	256	0.31	0.35	0.0	0.0	0.0	0.0
	500	1.18	1.33	0.0	0.0	0.0	0.0
Matrix	32	0.00	0.00	0.0	0.0	0.0	0.0
type	64	0.02	0.03	0.0	0.0	0.0	0.0
3	128	0.07	0.09	0.0	0.0	0.0	0.0
	256	0.30	0.34	0.0	0.0	0.0	0.0
	500	1.17	1.33	0.0	0.0	0.0	0.0
Matrix	32	0.01	0.00	0.0	0.0	0.0	0.0
type	64	0.02	0.02	0.0	0.0	0.0	0.0
4	128	0.07	0.08	0.0	0.0	0.0	0.0
	256	0.32	0.33	0.0	0.0	0.0	0.0
	500	1.17	1.34	0.0	0.0	0.0	0.0
Matrix	32	0.00	0.00	0.0	0.0	0.0	0.0
type	64	0.02	0.03	0.0	0.0	0.0	0.0
5	128	0.08	0.09	0.0	0.0	0.0	0.0
	256	0.30	0.34	0.0	0.0	0.0	0.0
	500	1.18	1.35	0.0	0.0	0.0	0.0
Matrix	32	0.01	0.00	0.0	0.0	0.0	0.0
type	64	0.02	0.02	0.0	0.0	0.0	0.0
6	128	0.07	0.10	0.0	0.0	0.0	0.0
	256	0.32	0.33	0.0	0.0	0.0	0.0
	500	1.17	1.32	0.0	0.0	0.0	0.0
Matrix	32	0.00	0.01	0.0	0.0	0.0	0.0
type	64	0.02	0.02	0.0	0.0	0.0	0.0
7	128	0.07	0.08	0.0	0.0	0.0	0.0
	256	0.32	0.33	0.0	0.0	0.0	0.0
	500	1.18	1.34	0.0	0.0	0.0	0.0
Matrix	32	0.79	1.01	2.13D−16	2.05D−16	1.39D−15	1.73D−15
type	64	2.67	3.99	1.73D−16	2.97D−16	3.38D−15	2.98D−15
8	128	11.33	15.36	6.41D−16	2.34D−16	3.84D−15	4.58D−15
	256	36.30	60.10	5.57D−16	3.87D−16	1.05D−14	1.52D−14
	500	142.8	226.6	6.38D−16	3.31D−16	2.02D−14	2.53D−14
Matrix	32	1.24	0.93	2.17D−16	1.03D−16	3.32D−16	4.44D−16
type	64	16.01	4.54	6.70D−17	1.03D−16	5.10D−15	4.94D−16
9	128	119.0	15.36	1.60D−16	1.40D−16	5.77D−15	7.77D−16
	256	937.8	154.5	1.14D−16	2.38D−16	1.19D−14	1.21D−15
	500	7334.	989.0	1.68D−16	3.87D−16	1.77D−14	2.10D−15
Matrix	32	1.31	0.80	3.76D−16	8.51D−17	8.32D−15	5.15D−15
type	64	22.23	3.91	4.34D−16	1.60D−16	5.91D−15	3.72D−15
10	128	171.4	24.22	7.32D−16	3.93D−16	3.83D−14	3.22D−14
	256	1432.	171.6	2.87D−16	5.81D−16	4.44D−14	1.29D−13
	500	12934	1193.	7.92D−16	6.56D−17	5.32D−14	5.22D−15

TABLE 7
Execution time (seconds) of computed eigenvalues and eigenvectors from type 11 *to* 21.

Matrix type	Order n	Execution time HOMO	Execution time B/III	$\max_i \|\|Ax_i - \lambda_i x_i\|\|_2/\lambda_{\max}$ HOMO	$\max_i \|\|Ax_i - \lambda_i x_i\|\|_2/\lambda_{\max}$ B/III	$\max_{i,j} \|(X^T X - I)_{i,j}\|/\lambda_{\max}$ HOMO	$\max_{i,j} \|(X^T X - I)_{i,j}\|/\lambda_{\max}$ B/III
Matrix type 11	32	0.71	1.00	1.83D−16	2.07D−16	7.75D−16	1.88D−15
	64	2.86	3.88	1.95D−16	2.35D−16	1.81D−15	5.93D−15
	128	11.14	24.28	5.08D−16	2.30D−16	2.63D−15	2.81D−14
	256	46.85	57.96	6.01D−16	2.90D−16	4.65D−15	1.44D−13
	500	174.3	217.4	1.04D−15	3.44D−16	1.15D−14	5.36D−13
Matrix type 12	32	0.53	1.07	2.26D−16	1.84D−16	3.20D−16	1.01D−16
	64	3.22	4.17	1.76D−16	2.51D−16	4.41D−16	8.44D−16
	128	12.23	14.78	6.82D−16	2.29D−16	1.55D−15	4.93D−16
	256	40.93	60.99	4.76D−16	2.51D−16	4.32D−15	5.73D−16
	500	159.3	229.1	1.24D−15	3.33D−16	1.46D−14	6.39D−15
Matrix type 13	32	0.67	0.96	3.71D−16	1.69D−16	1.72D−16	1.85D−16
	64	2.37	3.84	1.81D−16	2.69D−16	1.10D−16	1.79D−16
	128	9.95	14.98	8.93D−16	2.85D−16	3.04D−16	4.05D−16
	256	33.98	59.59	7.05D−16	2.21D−16	1.07D−15	7.07D−16
	500	157.9	223.4	9.38D−16	2.42D−16	1.29D−15	9.48D−16
Matrix type 14	32	0.52	0.99	8.47D−17	2.76D−16	1.45D−16	2.96D−16
	64	2.49	3.82	1.50D−16	1.81D−16	5.42D−16	1.40D−16
	128	9.70	14.77	2.40D−16	1.91D−16	1.23D−15	3.74D−15
	256	41.37	58.18	2.31D−16	2.66D−16	5.47D−15	8.71D−15
	500	137.4	217.3	1.17D−15	2.81D−16	1.67D−14	2.31D−14
Matrix type 15	32	0.80	1.08	1.63D−16	1.84D−16	1.44D−15	7.43D−16
	64	3.42	4.07	5.46D−16	2.63D−16	1.53D−15	5.86D−15
	128	9.97	15.35	1.73D−16	1.99D−16	3.77D−15	7.95D−15
	256	36.71	60.33	6.65D−16	2.16D−16	6.38D−15	4.48D−14
	500	143.8	229.7	8.55D−16	3.34D−16	2.74D−14	2.08D−14
Matrix type 16	32	0.54	1.03	8.96D−17	1.21D−16	3.33D−16	4.44D−16
	64	2.34	3.89	7.25D−17	2.01D−16	6.66D−16	7.77D−16
	128	9.96	15.15	9.94D−17	2.21D−16	2.80D−15	2.27D−15
	256	42.91	59.56	4.67D−16	1.82D−16	5.41D−15	5.64D−15
	500	150.5	227.3	1.14D−15	2.37D−16	1.70D−13	1.24D−14
Matrix type 17	32	2.08	0.97	6.46D−17	1.57D−16	2.55D−15	4.44D−16
	64	14.47	4.53	8.51D−17	9.18D−17	4.21D−15	1.10D−15
	128	105.4	24.58	9.34D−17	1.84D−16	7.99D−15	9.68D−16
	256	855.3	152.2	6.45D−17	1.57D−16	1.24D−14	3.00D−15
	500	9013	998.1	2.86D−16	1.58D−16	2.08D−14	3.20D−15
Matrix type 18	32	1.38	0.84	6.02D−16	1.31D−16	3.33D−16	7.45D−16
	64	23.17	3.85	2.38D−16	8.36D−17	2.87D−15	8.05D−16
	128	184.6	24.29	9.37D−16	1.29D−16	7.07D−14	1.60D−15
	256	1327.	170.4	1.88D−15	4.47D−16	5.31D−14	2.16D−13
	500	12814	1193.3	6.73D−15	3.71D−16	8.83D−14	1.32D−14
Matrix type 19	32	0.55	1.02	7.42D−17	1.53D−16	4.85D−16	7.16D−16
	64	2.39	3.81	2.59D−16	1.25D−16	7.10D−16	3.03D−16
	128	9.71	14.75	3.18D−16	1.83D−16	1.82D−15	1.14D−16
	256	45.70	57.56	8.97D−16	2.12D−16	3.58D−15	4.57D−15
	500	157.07	216.3	1.20D−15	2.19D−16	7.83D−15	1.65D−15
Matrix type 20	32	0.69	1.04	5.79D−16	1.26D−16	6.49D−16	3.09D−15
	64	3.73	3.88	1.46D−16	1.90D−16	7.30D−16	3.11D−15
	128	10.90	15.50	2.54D−16	1.65D−16	1.35D−15	2.45D−15
	256	42.95	60.29	4.45D−16	2.32D−16	3.02D−15	4.28D−15
	500	146.1	231.7	6.03D−16	2.20D−16	9.48D−15	3.91D−15
Matrix type 21	32	2.57	1.01	3.37D−17	1.00D−16	2.44D−15	6.66D−16
	64	14.10	4.96	4.33D−17	1.01D−16	3.99D−15	6.66D−16
	128	142.2	24.42	5.33D−17	1.15D−16	9.54D−15	6.66D−16
	256	1116.	150.7	6.25D−17	1.14D−16	1.24D−14	8.88D−16
	500	8702.	992.3	6.04D−17	1.14D−16	2.35D−14	1.75D−15

TABLE 8
Execution time (seconds), speedup, and efficiency of HOMO *on* [1,2,1] *and* T_2 *matrices.*

[1,2,1] matrix							T_2 matrix			
Order n	Nodes p	HOMO (Exe-Time)	S_P	$\frac{S_p}{p}$	TQL2 (Exe-Time)	TGL2 / HOMO	HOMO (Exe-Time)	S_p	$\frac{S_p}{p}$	TQL2 (Exe-) Time)
65	1	7.99	1.0	1.00	27.55	3.44	9.41	1.0	1.00	26.33
	2	4.51	1.8	0.89		6.10	4.93	1.9	0.95	
	4	2.77	2.9	0.72		9.93	2.88	3.3	0.82	
125	1	29.18	1.0	1.00	176.7	6.06	34.98	1.0	1.00	177.33
	2	15.25	1.9	0.96		11.59	17.55	2.0	1.00	
	4	8.76	3.3	0.83		20.18	9.34	3.7	0.94	
	8	6.38	4.6	0.57		27.70	6.14	5.7	0.71	
255	1	122.37	1.0	1.00	1457.	11.91	143.46	1.0	1.00	1497.
	2	69.93	1.9	0.97		20.85	71.76	2.0	1.00	
	4	33.08	3.7	0.92		44.07	37.2	3.9	0.96	
	8	20.38	6.0	0.75		71.53	22.4	6.4	0.80	
	16	14.28	8.6	0.54		102.08	14.85	9.7	0.60	
499	1	477.91	1.0	1.00	10889	22.79	550.90	1.0	1.00	11198
	2	242.01	2.0	0.99		45.00	276.56	2.0	1.00	
	4	125.45	3.8	0.95		86.80	140.48	3.9	0.98	
	8	74.49	6.4	0.80		146.19	81.72	6.7	0.84	
	16	47.41	10.1	0.63		229.69	52.27	10.5	0.66	

TABLE 9
Execution time (seconds), speedup, and efficiency of HOMO *on* $[1, \mu_i, 1]$ *and* BW *matrices.*

[1,μ_i,1] matrix					BW matrix				
Order n	Nodes p	HOMO (ExeTime)	S_p	$\frac{S_p}{p}$	Order n	Nodes p	HOMO (ExeTime)	S_p	$\frac{S_p}{p}$
65	1	7.61	1.0	1.00	64	1	11.89	1.0	1.00
	2	4.09	1.9	0.93		2	6.32	1.9	0.94
	4	2.81	2.7	0.68		4	3.79	3.1	0.78
125	1	30.21	1.0	1.00	128	1	26.87	1.0	1.00
	2	15.51	1.9	0.97		2	14.06	1.9	0.95
	4	9.50	3.2	0.79		4	7.69	3.5	0.87
	8	6.64	4.6	0.57		8	5.15	5.2	0.65
255	1	123.91	1.0	1.00	256	1	186.36	1.0	1.00
	2	62.73	2.0	0.98		2	97.75	1.9	0.97
	4	34.98	3.5	0.89		4	49.79	3.7	0.94
	8	20.86	5.9	0.74		8	31.55	5.9	0.74
	16	14.60	8.5	0.53		16	20.54	9.1	0.57
499	1	498.85	1.0	1.00	512	1	726.42	1.0	1.00
	2	249.46	2.0	1.00		2	366.40	2.0	0.99
	4	132.46	3.8	0.94		4	188.54	3.9	0.96
	8	78.18	6.4	0.80		8	111.51	6.5	0.81
	16	52.69	9.5	0.59		16	67.64	10.7	0.67

The results we show here are only preliminary. With further intensive concentrations on parallel programming strategies, our HOMO algorithm may be an excellent candidate for a variety of advanced architectures. We expect to report on this important aspect in a future article.

TABLE 10
Execution time (seconds), speedup, and efficiency of HOMO *on random matrices by computing the eigenvalues of the initial matrices with* TQL1.

Order	n =125				n =255				n =499			
Nodes	1	2	4	8	1	2	4	8	1	2	4	8
(ExeTime)	21.1	11.1	6.04	4.06	80.6	41.6	22.2	14.4	302.	155.	82.5	52.0
S_p	1.0	1.9	3.5	5.2	1.0	1.9	3.6	5.6	1.0	1.9	3.7	5.8
S_p/p	1.0	0.95	0.87	0.65	1.0	0.97	0.91	0.70	1.0	0.97	0.92	0.73

TABLE 11
Execution time (seconds), speedup, and efficiency of HOMO *on random matrices by computing the eigenvalues of the initial matrices with multisection method.*

Order	n =255					
Nodes	1	2	4	8	16	32
(ExeTime)	112.	57.6	30.2	16.7	10.1	7.07
S_p	1.0	2.0	3.7	6.7	11.1	16.0
S_p/p	1.0	0.98	0.93	0.84	0.69	0.50

Order	n =499						
Nodes	1	2	4	8	16	32	46
(ExeTime)	386.	195.	101.	53.5	31.1	19.5	16.8
S_p	1.0	2.0	3.8	7.2	12.4	19.7	22.9
S_p/p	1.0	0.99	0.95	0.90	0.78	0.62	0.50

REFERENCES

[1] M. T. CHU, *A simple application of the homotopy method to symmetric eigenvalue problems*, Linear Algebra Appl., 59 (1984), pp. 85–90.

[2] ———, *A note on the homotopy method for linear algebraic eigenvalue problems*, Linear Algebra Appl., 105 (1988), pp. 225–236.

[3] M. T. CHU, T. Y. LI, AND T. SAUER, *Homotopy method for general λ-matrix problems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 528–536.

[4] J. M. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–1951.

[5] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for symmetric eigenvalue problems*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139–s154.

[6] I. ILSE AND E. JESSUP, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 203–229.

[7] E. JESSUP, *Parallel solution of the symmetric tridiagonal eigenproblem*, Ph.D. thesis, Yale University, New Haven, CT, 1989.

[8] T. Y. LI AND N. H. RHEE, *Homotopy algorithm for symmetric eigenvalue problems*, Numer. Math., 55 (1989), pp. 256–280.

[9] T. Y. LI, T. SAUER, AND J. YORKE, *Numerical solution of a class of deficient polynomial systems*, SIAM J. Numer. Anal., 24 (1987), pp. 435–451.

[10] T. Y. LI, Z. ZENG, AND L. CONG, *Solving eigenvalue problems of real nonsymmetric matrices with real homotopy*, SIAM J. Numer. Anal., 29 (1992), pp. 229–248.

[11] T. Y. LI, H. ZHONG, AND X. H. SUN, *Parallel homotopy algorithm for symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 469–487.

[12] S. S. LO, B. PHILIPPE, AND A. SAMEH, *A multiprocessor algorithm for symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 155–165.

[13] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[14] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikeke, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines*—EISPACK *Guide*, 2nd ed., Springer-Verlag, New York, 1976.

[15] D. C. Sorensen and P. Tang, *On the orthogonality of eigenvectors computed by divide-and-conquer techniques*, preprint.

[16] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, New York, 1965.

SIAM J. SCI. COMPUT.
Vol. 14, No. 3, pp. 752–759, May 1993

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific 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.

THE DUAL SCHUR COMPLEMENT METHOD WITH WELL-POSED LOCAL NEUMANN PROBLEMS: REGULARIZATION WITH A PERTURBED LAGRANGIAN FORMULATION*

CHARBEL FARHAT†, PO-SHU CHEN†, AND FRANCOIS-XAVIER ROUX‡

Abstract. The dual Schur complement (DSC) domain decomposition (DD) method introduced by Farhat and Roux is an efficient and practical algorithm for the parallel solution of self-adjoint elliptic partial differential equations. A given spatial domain is partitioned into *disconnected* subdomains where an incomplete solution for the primary field is first evaluated using a direct method. Next, intersubdomain field continuity is enforced via a combination of discrete, polynomial, and/or piece-wise polynomial Lagrange multipliers, applied at the subdomain interfaces. This leads to a smaller size symmetric *dual* problem where the unknowns are the "gluing" Lagrange multipliers, and which is best solved with a preconditioned conjugate gradient (PCG) algorithm. However, for time-independent elasticity problems, every floating subdomain is associated with a singular stiffness matrix, so that the dual interface operator is in general indefinite. Previously, we have dealt with this issue by filtering out at each iteration of the PCG algorithm the contributions of the local null spaces. We have shown that for a small number of subdomains, say less than 32, this approach is computationally feasible. Unfortunately, the filtering phase couples the subdomain computations, increases the numerical complexity of the overall solution algorithm, and limits its *parallel implementation scalability*, and therefore is inappropriate for a large number of subdomains. In this paper, we regularize the DSC method with a perturbed Lagrangian formulation which restores the positiveness of the dual interface operator, reduces the computational complexity of the overall methodology, and improves its parallel implementation scalability. This regularization procedure corresponds to a novel splitting method of the interface operator which entails well-posed local discrete Neumann problems, even in the presence of floating subdomains. Therefore, it can also be interesting for other DD algorithms such as those considered by Bjorstad and Widlund, Marini and Quarteroni, De Roeck and Le Tallec, and recently by Mandel.

Key words. domain decomposition, Neumann problems, parallel processing

AMS(MOS) subject classifications. 65N20, 65N30, 65W5

1. Introduction.

Recently, Farhat and Roux have introduced a dual Schur complement (DSC) domain decomposition (DD) method for the efficient solution of static [1]–[4] and transient [5] finite element structural problems on parallel processors. The method was shown to outperform direct solvers on both serial and coarse-grained multiprocessors such as the CRAY Y-MP system, and to compare favorably with other domain decomposition algorithms on a 32-processor hypercube [2]. However, for a large number of subdomains and processors and for time-independent problems, the

*Received by the editors June 29, 1992; accepted for publication September 11, 1992. This work was supported by the NASA Langley Research Center under grant NAG-1536427, and by National Science Foundation grant ASC-8717773.

†Department of Aerospace Engineering Sciences and Center for Space Structures and Controls, University of Colorado at Boulder, Boulder, Colorado 80309-0429.

‡Office National d'Etudes et Recherches Aeronautiques, Groupe Calcul Parellèle, 29 Avenue de la Division Leclerc, BP72 92322 Chatillon Cedex, France.

DSC method may lose some of its efficiency because of its special treatment of floating subdomains. The objective of this paper is to present a regularization procedure that is based on a "balanced" perturbed Lagrangian formulation and that improves the overall computational efficiency of the DSC method. This regularization procedure corresponds to a novel splitting method of the interface operator which entails well-posed local discrete Neumann problems, even in the presence of floating subdomains. It provides a practical means to regularize several other DD algorithms using both Dirichlet and Neumann local solvers and which can suffer from the presence of floating subdomains [6]–[9].

The resulting regularized dual Schur complement (RDSC) method is a two-field alternative to the related three-field hybrid domain decomposition method introduced by Glowinski and Le Tallec [10]. Because of space limitations, we refer the reader to [2] and [3] for a background on the DSC method and for a discussion on the effect of floating subdomains on the algebraic properties and computational requirements of the resulting interface problem.

2. A regularized DSC method.

2.1. The two-subdomain problem. The variational form of the three-dimensional boundary-value problem to be solved is as follows. Given f and h, find the displacement function u that is a stationary point of the energy functional

$$J(v) = \tfrac{1}{2}a(v,v) - (v,f) - (v,h)_\Gamma \ , \tag{2.1}$$

where

$$a(v,w) = \int_\Omega v_{(i,j)} c_{ijkl} w_{(k,l)}\, d\Omega; \quad (v,f) = \int_\Omega v_i f_i \, d\Omega; \quad (v,h)_\Gamma = \int_{\Gamma_h} v_i h_i \, d\Gamma.$$

In the above, the indices i, j, k take the values 1–3, $v_{(i,j)} = (v_{i,j} + v_{j,i})/2$, $v_{i,j}$ denotes the partial derivative of the ith component of v with respect to the jth spatial variable, c_{ijkl} are the elastic coefficients, Ω denotes the volume of the elastostatic body, Γ its piecewise smooth boundary, and Γ_h is the piece of Γ where the tractions h_i are prescribed. If Ω is subdivided into two subdomains Ω_1 and Ω_2, solving the above elastostatic problem is equivalent to finding the displacement functions u_1 and u_2, which are stationary points of the perturbed Lagrangian functional

$$\begin{aligned} H(v_1, v_2, \lambda) &= J_1(v_1) + J_2(v_2) + \textstyle\int_{\Gamma_I} \lambda(v_1 - v_2)\, d\Gamma \\ &\quad -\tfrac{1}{2} \textstyle\int_{\overline{\Gamma}_I} \mathcal{L}(v_1|_{\overline{\Gamma}_I})^2 - \mathcal{L}(v_2|_{\overline{\Gamma}_I})^2 d\Gamma, \end{aligned} \tag{2.2}$$

where $\overline{\Gamma}_I \subset \Gamma_I$ is a subset of the interface boundary, and $\mathcal{L}$ is a linear operator that acts on the traces of v_1 and v_2 on $\overline{\Gamma}_I$. The proper selection of $\overline{\Gamma}_I$ and $\mathcal{L}$ is discussed later. In the remainder of this section, we assume that only Ω_2 is a floating subdomain, that is, a subdomain without sufficient Dirichlet boundary conditions to guarantee a nonsingular stiffness matrix. The finite element equations associated with (2.2) are given by

$$(\mathbf{K}_1 - \boldsymbol{\Delta})\mathbf{u}_1 = \mathbf{f}_1 - \mathbf{B}_1^T\boldsymbol{\lambda}; \quad (\mathbf{K}_2 + \boldsymbol{\Delta})\mathbf{u}_2 = \mathbf{f}_2 - \mathbf{B}_2^T\boldsymbol{\lambda}; \quad \mathbf{B}_1\mathbf{u}_1 + \mathbf{B}_2\mathbf{u}_2 = 0, \tag{2.3}$$

where $\mathbf{K}_1$ is positive definite, $\mathbf{K}_2$ is positive semidefinite, and $\boldsymbol{\Delta}$ is zero everywhere except on a subset of the interface boundary degrees of freedom. If discrete Lagrange multipliers are used, $\mathbf{B}_1$ and $\mathbf{B}_2$ are signed boolean matrices; otherwise, they are standard finite element matrices with full column rank. Clearly, $\mathbf{\Delta}$ serves the purpose of regularizing $\mathbf{K}_2$. However, $\mathbf{\Delta}$ is useful only if it is sparse and computationally inexpensive, and if it restores the positive definiteness nature of the stiffness in Ω_2, without destroying this algebraic property in Ω_1. In other words, the problem now is to find an economical matrix $\mathbf{\Delta}$—or its corresponding operator $\mathcal{L}$—which can stiffen Ω_2 enough so that $\mathbf{K}_2+\mathbf{\Delta}$ is symmetric positive definite, without softening Ω_1 enough so that $\mathbf{K}_1-\mathbf{\Delta}$ is singular or indefinite. We refer to this problem as a *balancing* problem, and we refer to the corresponding functional $H(v_1, v_2, \lambda)$ as a *balanced* perturbed Lagrangian.

Remark 2.1.1. Given that $\int_{\overline{\Gamma}_I} \mathcal{L}(v_1|_{\overline{\Gamma}_I})^2 - \mathcal{L}(v_2|_{\overline{\Gamma}_I})^2 \, d\Gamma$ depends only on the traces of v_1 and v_2 on the subset $\overline{\Gamma}_I$ of the interface boundary Γ_I where the Lagrange multipliers λ enforce the continuity equation $(v_1 - v_2)|_{\Gamma_I} = 0$, the solution $(\mathbf{u}_1, \mathbf{u}_2)$ of the system of equations (2.3) is independent of the operator $\mathcal{L}$ and its corresponding matrix $\mathbf{\Delta}$.

2.2. Balancing the subdomains. Here we consider the problem of constructing $\mathbf{\Delta}$ such that both $\mathbf{K}_1 - \mathbf{\Delta}$ and $\mathbf{K}_2 + \mathbf{\Delta}$ are symmetric positive definite. Homogeneous boundary conditions are assumed on part of Ω_1 (Fig. 1).

Let n_I and n_r denote, respectively, the total number of unknowns on Γ_I, and the exact number of rigid body modes associated with Ω_2, that is, the dimension of the null space of $\mathbf{K}_2$. These rigid body modes can be eliminated, for example, by fixing n_c degrees of freedom on Γ_I, where $n_r \leq n_c \leq n_I$. We define $\overline{\Gamma}_I$ as the spatial support of these n_c degrees of freedom (Fig. 1).

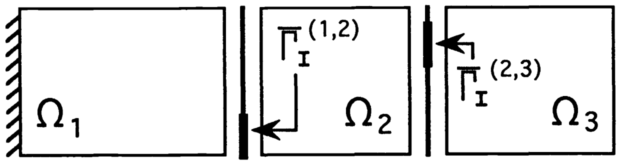


FIG. 1. $\overline{\Gamma}_I$ *for the two-subdomain problem with a local singularity.*

Next, we partition the subdomain stiffness matrices as

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{K}_1^{ff} & \mathbf{K}_1^{fc} \\ \mathbf{K}_1^{fc^T} & \mathbf{K}_1^{cc} \end{bmatrix} \quad \text{and} \quad \mathbf{K}_2 = \begin{bmatrix} \mathbf{K}_2^{ff} & \mathbf{K}_2^{fc} \\ \mathbf{K}_2^{fc^T} & \mathbf{K}_2^{cc} \end{bmatrix}, \tag{2.4}$$

where the superscript ^{c} denotes those degrees of freedom on $\overline{\Gamma}_I$, and the superscript ^{f} denotes all of the other degrees of freedom. The above partitioning is characterized by:

1. $\mathbf{K}_1^{ff}$ and $\mathbf{K}_2^{ff}$ are symmetric positive definite,
2. $\mathbf{K}_1^{fc}$ and $\mathbf{K}_2^{fc}$ have full column rank,
3. $\mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T} \mathbf{K}_1^{ff^{-1}} \mathbf{K}_1^{fc}$ is symmetric positive definite, and
4. $\mathbf{K}_2^{cc} - \mathbf{K}_2^{fc^T} \mathbf{K}_2^{ff^{-1}} \mathbf{K}_2^{fc}$ is symmetric positive semidefinite.

After all but the last n_c equations in each subdomain are reduced with a Gaussian elimination or a Cholesky decomposition, $\mathbf{K}_1$ and $\mathbf{K}_2$ are overwritten with

$$\mathbf{K}_1^R = \begin{bmatrix} \mathbf{K}_1^{ff^R} & \mathbf{K}_1^{fc^R} \\ \cdots & \mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T}\mathbf{K}_1^{ff^{-1}}\mathbf{K}_1^{fc} \end{bmatrix}, \qquad (2.5)$$
$$\mathbf{K}_2^R = \begin{bmatrix} \mathbf{K}_2^{ff^R} & \mathbf{K}_2^{fc^R} \\ \cdots & \mathbf{K}_2^{cc} - \mathbf{K}_2^{fc^T}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc} \end{bmatrix},$$

where the superscript ^{R} indicates a reduced matrix. Now, if $\mathbf{\Delta}$ is constructed as

$$\mathbf{\Delta} = \mathbf{\Delta}^* = \frac{1}{2}(\mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T}\mathbf{K}_1^{ff^{-1}}\mathbf{K}_1^{fc}), \qquad (2.6)$$

then it follows that

$$(\mathbf{K}_1 - \mathbf{\Delta}^*)^R = \begin{bmatrix} \mathbf{K}_1^{ff^R} & \mathbf{K}_1^{fc^R} \\ \cdots & \frac{1}{2}(\mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T}\mathbf{K}_1^{ff^{-1}}\mathbf{K}_1^{fc})^R \end{bmatrix}, \qquad (2.7)$$
$$(\mathbf{K}_2 + \mathbf{\Delta}^*)^R = \begin{bmatrix} \mathbf{K}_2^{ff^R} & \mathbf{K}_2^{fc^R} \\ \cdots & [\mathbf{K}_2^{cc} - \mathbf{K}_2^{fc^T}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc} + \frac{1}{2}(\mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T}\mathbf{K}_1^{ff^{-1}}\mathbf{K}_1^{fc})]^R \end{bmatrix},$$

which clearly demonstrates that both $\mathbf{K}_1 - \mathbf{\Delta}^*$ and $\mathbf{K}_2 + \mathbf{\Delta}^*$ are symmetric positive definite matrices.

The extreme values of n_c correspond to two extreme regularization strategies. For $n_c = n_I$, the local problems are better conditioned than for $n_c = n_r$, but a lot of fill-in may be introduced during the factorization of $\mathbf{K}_2 + \mathbf{\Delta}^*$. On the other hand, for $n_c = n_r$, the local problems are not as well conditioned as for $n_c = n_I$, but $\mathbf{K}_2 + \mathbf{\Delta}^*$ and $\mathbf{K}_2$ generally have the same sparsity pattern, and therefore the latter regularization strategy is cheaper to implement. The case $n_r \leq n_c \leq n_I$ corresponds to a compromise between the two extreme options. In the remainder of this paper, we consider only the case where $n_c = n_r$, and therefore we have

$$\mathbf{K}_2^{cc} - \mathbf{K}_2^{fc^T}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc} = 0. \qquad (2.8)$$

2.3. The multiple subdomain problem. For the sake of clarity, we first consider a strip-wise decomposed three-subdomain problem where Ω_1 is clamped at one end, and Ω_2 and Ω_3 are floating subdomains (Fig. 2). $\overline{\Gamma}_I^{(1,2)}$ is defined as the spatial support of $n_r^{(1,2)}$ degrees of freedom on the interface between Ω_1 and Ω_2 that must be constrained in order to remove the rigid body modes in Ω_2. Similarly, $\overline{\Gamma}_I^{(2,3)}$ is defined as the spatial support of $n_r^{(2,3)}$ degrees of freedom on the interface between Ω_2 and Ω_3 which must be constrained in order to eliminate the rigid body modes in Ω_3, *after* Ω_2 has been regularized. For subdomain Ω_2, we refer to the degrees of freedom on $\overline{\Gamma}_I^{(1,2)}$ as the *receiving* degrees of freedom, and to those on $\overline{\Gamma}_I^{(2,3)}$ as the *emitting* degrees of freedom.

We partition the subdomain stiffness matrices as

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{K}_1^{ff} & \mathbf{K}_1^{fc} \\ \mathbf{K}_1^{fc^T} & \mathbf{K}_1^{cc} \end{bmatrix}; \mathbf{K}_2 = \begin{bmatrix} \mathbf{K}_2^{ff} & \mathbf{K}_2^{fc^r} & \mathbf{K}_2^{fc^e} \\ \mathbf{K}_2^{fc^{rT}} & \mathbf{K}_2^{c^rc^r} & \mathbf{K}_2^{c^rc^e} \\ \mathbf{K}_2^{fc^{eT}} & \mathbf{K}_2^{c^rc^{eT}} & \mathbf{K}_2^{c^ec^e} \end{bmatrix}; \mathbf{K}_3 = \begin{bmatrix} \mathbf{K}_3^{ff} & \mathbf{K}_3^{fc} \\ \mathbf{K}_3^{fc^T} & \mathbf{K}_3^{cc} \end{bmatrix}, \qquad (2.9)$$

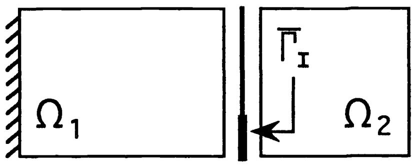


FIG. 2. *The strip-wise decomposed three-subdomain problem with local singularities.*

where the superscripts $^{c^r}$ and $^{c^e}$ indicate the receiving and emitting degrees of freedom in Ω_2, respectively. After all but the last $n_r^{(1,2)}$ equations in Ω_1 and Ω_2, and all but the last $n_r^{(2,3)}$ equations in Ω_3 are reduced, the subdomain stiffness matrices are overwritten with

(2.10)
$$\mathbf{K}_1^R = \begin{bmatrix} \mathbf{K}_1^{ff^R} & \mathbf{K}_1^{fc^R} \\ \cdots & \mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T}\mathbf{K}_1^{ff^{-1}}\mathbf{K}_1^{fc} \end{bmatrix}, \qquad \mathbf{K}_3 = \begin{bmatrix} \mathbf{K}_3^{ff^R} & \mathbf{K}_3^{fc^R} \\ \cdots & \mathbf{O} \end{bmatrix}$$

and

$$\mathbf{K}_2^R = \begin{bmatrix} \mathbf{K}_2^{ff^R} & \mathbf{K}_2^{fc^{r}R} & \mathbf{K}_2^{fc^{e}R} \\ \cdots & \mathbf{K}_2^{c^rc^r} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^r} & \mathbf{K}_2^{c^rc^e} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e} \\ \cdots & [\mathbf{K}_2^{c^rc^e} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e}]^T & \mathbf{K}_2^{c^ec^e} - \mathbf{K}_2^{fc^{eT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e} \end{bmatrix}.$$

First, Ω_2 is regularized with the $n_r^{(1,2)} \times n_r^{(1,2)}$ matrix

$$\mathbf{\Delta}^{(1,2)} = \frac{1}{2}(\mathbf{K}_1^{cc} - \mathbf{K}_1^{fc^T}\mathbf{K}_1^{ff^{-1}}\mathbf{K}_1^{fc}),$$

after which the factored stiffnesses in Ω_1 and Ω_2 become

$$(\mathbf{K}_1 - \mathbf{\Delta}^{(1,2)})^R = \begin{bmatrix} \mathbf{K}_1^{ff^R} & \mathbf{K}_1^{fc^R} \\ \cdots & \mathbf{\Delta}^{(1,2)^R} \end{bmatrix} \tag{2.11}$$

and

(2.12)
$$(\mathbf{K}_2 + \mathbf{\Delta}^{(1,2)})^R =$$
$$\begin{bmatrix} \mathbf{K}_2^{ff^R} & \mathbf{K}_2^{fc^{r}R} & \mathbf{K}_2^{fc^{e}R} \\ \cdots & [\mathbf{K}_2^{c^rc^r} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^r} + \mathbf{\Delta}^{(1,2)}]^R & [\mathbf{K}_2^{c^rc^e} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e}]^R \\ \cdots & \cdots & \mathbf{T}_2 \end{bmatrix},$$

where

$$\mathbf{T}_2 = \mathbf{K}_2^{c^ec^e} - \mathbf{K}_2^{fc^{eT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e} - [\mathbf{K}_2^{c^rc^e} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e}]^T$$
$$[\mathbf{K}_2^{c^rc^r} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^r} + \mathbf{\Delta}^{(1,2)}]^{-1}[\mathbf{K}_2^{c^rc^e} - \mathbf{K}_2^{fc^{rT}}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e}].$$

Next, Ω_3 is regularized with the $n_r^{(2,3)} \times n_r^{(2,3)}$ matrix $\mathbf{\Delta}^{(2,3)} = \frac{1}{2}\mathbf{T}_2$ and finally the factored stiffnesses in Ω_2 and Ω_3 become

$$\text{(2.13)} \quad (\mathbf{K}_2 - \mathbf{\Delta}^{(2,3)})^R =$$

$$\begin{bmatrix} \mathbf{K}_2^{ff^R} & \mathbf{K}_2^{fc^{r}R} & \mathbf{K}_2^{fc^{e}R} \\ \cdots & [\mathbf{K}_2^{c^r c^r} - \mathbf{K}_2^{fc^{r}T}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^r} + \mathbf{\Delta}^{(1,2)}]^R & [\mathbf{K}_2^{c^r c^e} - \mathbf{K}_2^{fc^{r}T}\mathbf{K}_2^{ff^{-1}}\mathbf{K}_2^{fc^e}]^R \\ \cdots & \cdots & \mathbf{\Delta}^{(2,3)^R} \end{bmatrix}$$

and

$$\text{(2.14)} \qquad (\mathbf{K}_3 + \mathbf{\Delta}^{(2,3)})^R = \begin{bmatrix} \mathbf{K}_3^{ff^R} & \mathbf{K}_3^{fc^R} \\ \cdots & \mathbf{\Delta}^{(2,3)^R} \end{bmatrix}.$$

For arbitrary mesh decompositions, the crucial point is to detect for a given subdomain Ω_s and a neighboring Ω_q, whether $\overline{\Gamma}_I^{(s,q)}$ is a "receiving" or "emitting" subset of Γ_I. If V_s denote the number of neighboring subdomains to Ω_s, the overall regularization algorithm can be implemented as follows:

Step 1. For every subdomain Ω_s, reduce all but the last $n_r^{(s)}$ equations of the corresponding stiffness matrix $\mathbf{K}_s$. (Note that $n_r^{(s)} \leq 3 \times V_s$ in two-dimensional problems, and $n_r^{(s)} \leq 6 \times V_s$ in three-dimensional ones.)

Step 2. Regularization proceeds from the nonfloating subdomains towards the floating ones. Floating subdomains which contain both "receiving" and "emitting" interface subsets are regularized first, and the "receiving" degrees of freedom are treated before the "emitting" ones.

Clearly Step 1 is a parallel step but Step 2 is a sequential one. However, for most large-scale problems, the CPU time corresponding to Step 2 is an extremely small fraction of the CPU time corresponding to Step 1, as the first step may involve millions of floating-point operations while the second one involves at most a few hundred. Interprocessor communication is required only in Step 2 and is limited to neighboring processors.

3. The regularized interface problem. If the floating subdomains are not regularized, the interface problem associated with (2.3) can be written in the case of N_s arbitrary subdomains as [1]

$$\text{(3.1)} \qquad \begin{bmatrix} \sum_{s=1}^{s=N_s} \mathbf{B}_s\mathbf{K}_s^*\mathbf{B}_s^T & -\mathbf{G}_I \\ -\mathbf{G}_I{}^T & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ -\mathbf{e} \end{bmatrix}, \quad \text{where}$$
$$\mathbf{K}_s^* = \mathbf{K}_s^{-1} \text{ if } \Omega_s \text{ is not a floating subdomain,}$$
$$\mathbf{K}_s^* = \mathbf{K}_s^{+} \text{ if } \Omega_s \text{ is a floating subdomain,}$$

where $\mathbf{K}_s^+$ is a generalized inverse of $\mathbf{K}_s$, $\mathbf{G}_I$ is a full column rank matrix that stores the traces on the interface boundary of the null spaces corresponding to the floating subdomains, and $\boldsymbol{\alpha}$ stores the contributions of these null spaces to the local solutions. The above interface problem (3.1) is clearly indefinite and its solution requires special handling [1]–[3]. On the other hand, the regularized interface problem associated with (2.3) can be written for N_s arbitrary subdomains as

$$\left[\sum_{s=1}^{s=N_s} \mathbf{B}_s(\mathbf{K}_s + \delta_s\boldsymbol{\Delta}_s)^{-1}\mathbf{B}_s^T\right]\boldsymbol{\lambda} = \sum_{s=1}^{s=N_s} \mathbf{B}_s(\mathbf{K}_s + \delta_s\boldsymbol{\Delta}_s)^{-1}\mathbf{f}_s, \tag{3.2}$$

where δ_s is $+1$, -1, or 0 if Ω_s is a receiving, sending, or nonfloating subdomain, respectively. Clearly, the above regularized system (3.2) is symmetric positive definite. The PCG algorithm can be applied to the solution of (3.2) at little additional cost and using only subdomain-by-subdomain scalable and intrinsically parallel computations. As in the case of nonfloating subdomains, the above interface problem is preconditioned with $\mathbf{P}$ derived in [1] as

$$\mathbf{P}^{-1} = \sum_{s=1}^{s=N_s} \mathbf{B}_s\mathbf{K}_s\mathbf{B}_s^T. \tag{3.3}$$

4. Performance improvement. Here, we consider the static analysis of the cabin of a launch vehicle subjected to external aerodynamic loading and internal pressurization (Fig. 3). All computations are performed on an iPSC/860 Touchstone machine. The structure is discretized with triangular shell elements. Several meshes of different sizes are constructed, each corresponding to a different number of processors and therefore to a different total memory size. In order to highlight the improvement in performance induced by the regularization approach described in this paper, both the "filtered" DSC method [1] and the RDSC method are applied to the solution of the same problem. Also, the Jacobi preconditioned (diagonal scaling) conjugate gradient (JPCG) algorithm is used as a reference for CPU timings.

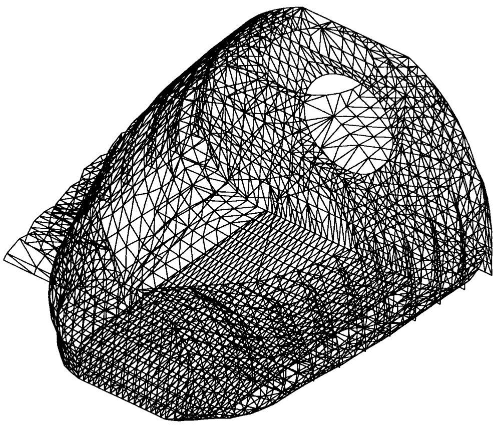


FIG. 3. *Finite element discretization of the cabin.*

All measured performance results are summarized in Table 1, where NP, NDOF, and T_p denote, respectively, the number of processors, the number of degrees of freedom, and the solution parallel time measured in seconds. The number of iterations is indicated in parentheses. For all solution algorithms, convergence is established by requiring that the normalized global residual be less than 10^{-3}:

$$\frac{||\mathbf{Ku} - \mathbf{f}||_2}{||\mathbf{f}||_2} \leq 10^{-3}. \tag{4.1}$$

TABLE 1
Performance results—iPSC/860.

NP	NDOF	T_p (JPCG)	T_p (DSC)	T_p (RDSC)
4	4,024	46 s. (700 it.)	23 s. (34 it.)	22 s. (38 it.)
16	16,456	110 s. (2308 it.)	54 s. (66 it.)	49 s. (62 it.)
32	31,928	168 s. (3980 it.)	79 s. (89 it.)	62 s. (92 it.)
64	59,064	295 s. (6127 it.)	189 s. (142 it.)	121 s. (144 it.)

The performance results reported above clearly indicate that

1. the regularization process does not seem to negatively affect the performance of the solution of the interface problem, since the DSC and RDSC method appear to converge after an almost identical number of iterations,
2. as expected, the performance improvement due to regularization is most important for an increasing number of processors, and
3. both DSC methods are shown to outperform the JPCG algorithm by a factor of three.

REFERENCES

[1] C. Farhat and F. X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, Internat. J. Numer. Methods Engrg., 32 (1991), pp. 1205–1227.

[2] ———, *An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 379–396.

[3] C. Farhat, *A Lagrange multiplier based divide and conquer finite element algorithm*, J. Comput. Systems Engrg., 2 (1991), pp. 149–156.

[4] ———, *A saddle-point principle domain decomposition method for the solution of solid mechanics problems*, in Proc. Fifth SIAM Conf. on Domain Decomposition Methods for Partial Differential Equations, Society for Industrial and Applied Mathematics, Phila., PA, 1991.

[5] ———, *Parallel processing in structural mechanics: Blending mathematical, implementational, and technological advances*, in Computing Methods in Applied Sciences and Engineering, R. Glowinski, ed., Nova Science Publishers, Inc., New York, 1992, pp. 89–106.

[6] P.E. Bjorstad and O.B. Widlund, *Iterative methods for solving elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1097–1120.

[7] L.D. Marini and A. Quarteroni, *An iterative procedure for domain decomposition methods: A finite element approach*, in Proc. First Internat. Sympos. on Domain Decomposition Methods for Partial Differential Equations, Society for Industrial and Applied Mathematics, Phila., PA, 1988.

[8] Y.H. De Roeck and P. Le Tallec, *Analysis and test of a local domain decomposition preconditioner*, in Proc. Fourth SIAM Internat. Conf. on Domain Decomposition Methods, Society for Industrial and Applied Mathematics, Phila., PA, 1991.

[9] J. Mandel, *Balancing domain decomposition*, Comm. Appl. Numer. Meth., to appear.

[10] R. Glowinski and P. Le Tallec, *Augmented Lagrangian interpretation of the nonoverlapping Schwarz alternating method*, in Proc. Third Internat. Sympos. on Domain Decomposition Methods for Partial Differential Equations, Society for Industrial and Applied Mathematics, Phila., PA, 1989, pp. 224-231.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 761–769, July 1993

A SUPERNODAL CHOLESKY FACTORIZATION ALGORITHM FOR SHARED-MEMORY MULTIPROCESSORS*

ESMOND NG† AND BARRY W. PEYTON†

Abstract. This paper presents a parallel sparse Cholesky factorization algorithm for shared-memory MIMD multiprocessors. The algorithm is particularly well suited for vector supercomputers with multiple processors, such as the Cray Y-MP. The new algorithm is a straightforward parallelization of the left-looking supernodal sparse Cholesky factorization algorithm. Like its sequential predecessor, it improves performance by reducing indirect addressing and memory traffic. Experimental results on a Cray Y-MP demonstrate the effectiveness of the new algorithm. On eight processors of a Cray Y-MP, the new routine performs the factorization at rates exceeding one Gflop for several test problems from the Harwell–Boeing sparse matrix collection.

Key words. parallel algorithms, sparse linear systems, Cholesky factorization, supernodes

AMS subject classifications. 65F, 65W

1. Introduction.

In this paper, we present a simple algorithm for computing a sparse Cholesky factorization on a shared-memory multiprocessor system. While our focus is on powerful multiprocessor vector supercomputers, our approach should perform well on any shared-memory multiprocessor with a modest number of processors and a bus that provides adequate bandwidth. The key features of our algorithm can be summarized as follows.

- It is both column oriented and left looking, like the algorithms used in several sparse matrix packages (e.g., see [11]).
- It uses supernodes to reduce both memory traffic and indirect addressing, as described in [4].
- The techniques used to implement the algorithm on a shared-memory multiprocessor are precisely those used in [10] to implement the left-looking algorithm in [11] on this class of machines.

Our primary contribution is to incorporate into one algorithm the supernodal techniques introduced in [4] and the parallel implementation techniques introduced in [10], and further to demonstrate the effectiveness of this simple approach on an important class of machines. Since the ideas upon which the algorithm is based are simple and well known, our discussion of previous work and the presentation of the new algorithm will be brief and, for the most part, informal. The reader can find most of the necessary details in [4], [10], and [11].

This note is organized as follows. Section 2 briefly discusses some background material. Section 3 provides a complete description of our algorithm. The timing results reported in §4 demonstrate the effectiveness of this approach on a Cray Y-MP with eight processors. Finally, §5 contains a few concluding remarks.

2. Background.

Let A be a symmetric positive definite matrix. The Cholesky factor of A, denoted by L, is a lower triangular matrix whose main diagonal is positive and for

*Received by the editors April 29, 1991; accepted for publication (in revised form) September 11, 1992. This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

†Mathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, Tennessee 37831-6367.

which $A = LL^T$. When A is sparse, fill occurs during the factorization; that is, some of the zero elements in A will become nonzero elements in L. In order to reduce time and storage requirements, only the nonzero positions of L are stored and operated on during *sparse* Cholesky factorization. Techniques for accomplishing this task and for reducing fill have been studied extensively (see [11] for details). In this paper we restrict our attention to the numerical factorization phase. We assume that the preprocessing steps, such as reordering to reduce fill and symbolic factorization to set up the compact data structure for L, have been performed. Details on the preprocessing can also be found in [11].

Column-oriented, left-looking, sparse Cholesky. The standard column-oriented, left-looking, sparse Cholesky algorithm is widely used in many sparse matrix packages, such as SPARSPAK [5]. When column $L_{*,j}$ is to be computed, the algorithm modifies column $A_{*,j}$ with multiples of previous columns of L, namely those columns $L_{*,k}$ for which $1 \leq k \leq j-1$ and $L_{j,k} \neq 0$. We shall let *cmod*(j, k) denote the operation of updating column $A_{*,j}$ with a multiple of the appropriate entries of $L_{*,k}$. The compute-intensive kernel used by the algorithm performs a single *cmod*(j, k) operation. A detailed description of this algorithm, along with a computer code that implements it, can be found in [11].

Supernodal sparse Cholesky. For many realistic problems, especially those from structural analysis applications, the fill generated during the factorization creates groups of contiguous columns that share the "same" sparsity structure. A group of such columns is referred to as a *supernode*. The reader should consult [12] and [16] for a detailed description of various *supernode partitions*. The first reference also presents an efficient algorithm for generating a supernode partition.

Recently, supernodes have been used to organize sparse numerical factorization algorithms around matrix-vector or matrix-matrix operations that reduce memory traffic, thereby making more efficient use of vector registers [3], [4] or cache [1], [17]. The role of supernodes in improving both left- and right-looking sparse Cholesky factorization algorithms is well documented [1], [3], [4], [8], [17], [19], with the references [4] and [17] providing the basis for the work in this note.

Let $\boldsymbol{K} = \{p, p+1, \ldots, p+q\}$ denote a supernode in L. Because of the sparsity pattern shared by all columns in $\boldsymbol{K}$, any column $A_{*,j}$ $(j > p+q)$ will be modified by either *all* columns of $\boldsymbol{K}$ or *no* column of $\boldsymbol{K}$. Consequently, a supernodal Cholesky algorithm is expressed in terms of the operation *cmod*$(j, \boldsymbol{K})$ $(j \notin \boldsymbol{K})$, which modifies column j with a multiple of each column in $\boldsymbol{K}$, and also the operation *cmod*$(j, \boldsymbol{J})$ $(j \in \boldsymbol{J})$, which modifies column j with a multiple of each column $k \in \boldsymbol{J}$ for which $k < j$. The *cmod*$(j, \boldsymbol{K})$ update, where $j \notin \boldsymbol{K}$, first uses a *dense matrix-vector multiplication* to accumulate the column modifications in a small work vector, after which the accumulated column update is applied to the target column $L_{*,j}$ using a single column operation that requires indirect addressing. In practice, this often greatly reduces the number of operations that require indirect addressing. Execution of the *cmod*$(j, \boldsymbol{J})$ update, where $j \in \boldsymbol{J}$, is even easier; the result of the matrix-vector multiplication is accumulated directly into factor storage for the target column $L_{*,j}$ and thus requires no work storage or indirect addressing. For both *cmod*$(j, \boldsymbol{K})$ and *cmod*$(j, \boldsymbol{J})$, unrolling the outer loop of the matrix-vector multiplication significantly reduces memory traffic, thereby improving the utilization of pipelined arithmetic units, especially on vector supercomputers. The interested reader should consult [4] and [17] for details.

Automatic parallelization of the supernodal algorithm. One approach to implementing sparse supernodal Cholesky factorization on a multiprocessor vector supercomputer has been described in [19]. The basic idea is to partition the work in *cmod*$(j, \boldsymbol{K})$ and *cmod*$(j, \boldsymbol{J})$ evenly among the available processors using the automatic parallelization capabilities of the target machine. This approach is similar to that employed in the LAPACK project [2], where, in the interest of software portability and reliability, use of multiple processors occurs *strictly within* each call to some compute-intensive variant of a matrix-matrix multiplication (BLAS3) or matrix-vector multiplication (BLAS2) kernel subroutine. Hence each call to the kernel involves a fork-and-join operation. For large dense matrices, where the vectors are quite long and each call to the kernel routine typically involves a substantial amount of work, this approach is quite effective [6]. For sparse matrices, however, short vectors and a limited amount of work within a typical call to the kernel routine make it quite difficult to implement this approach in an effective manner. As shall be shown in §4, the performance of the code in [19] apparently suffers from these defects.

Hand parallelization of the "nodal" algorithm. George, Heath, Liu, and Ng [10] introduced a parallel sparse Cholesky algorithm for shared-memory multiprocessors. Their algorithm is a straightforward parallel implementation of the sequential column-based, left-looking, sparse Cholesky factorization algorithm from [11]. The techniques we use to parallelize the supernodal factorization algorithm are taken directly from [10].

The task of computing column $L_{*,j}$ is referred to as a *column task* in the computation and is denoted by $Tcol(j)$. More precisely,

$$Tcol(j) := \{cmod(j,k) \mid k < j \text{ and } L_{j,k} \neq 0\} \cup \{cdiv(j)\},$$

where $cdiv(j)$ is the column scaling required to complete the computation of $L_{*,j}$. The parallel algorithm maintains a pool of column tasks, and the assignment of column tasks to processors is dynamic: when a processor completes one column task, it immediately seeks another column task from the pool. This approach is particularly appropriate for a shared-memory multiprocessor system with a fast bus, and, as might be expected, it typically achieves good load balance.

Clearly each task $Tcol(j)$ needs access to the set of column indices that constitute the sparsity pattern of row $L_{j,*}$. Because the computation and its data structures are both column oriented, special techniques are needed to generate these row-oriented sets in all the left-looking algorithms. The details of how to use a collection of linked lists to compute the "row structure" sets as the algorithm proceeds are well known and can be found in [11]. These linked lists are implemented with a single n-vector. To maintain the integrity of this dynamic data structure, it is modified in critical sections of the code, which are implemented by means of standard synchronizing primitives, which we shall call `lock` and `unlock`.

Finally, the column tasks $Tcol(j)$ are placed in the column-task queue in an order that exploits high-level parallelism in the computation. It is well known that the elimination tree associated with L fully describes the data dependencies in the computation [14]. We refer the reader to [15] for details on the many uses of elimination trees in sparse matrix computation, and to [14] for details on how the elimination tree reveals various levels of parallelism available in the factorization. Two columns are independent of one another if and only if the corresponding nodes in the elimination tree are not an ancestor-descendent pair in the tree. Consequently, an ordering obtained by a breadth-first, bottom-up traversal of the postordered elimination tree is ideal for preserving as much independence as possible among the columns currently being computed at any point in the algorithm.

3. Detailed algorithm.

In this section we give the details of an algorithm based on the ideas discussed in §2. The following notation and conventions are used in the algorithm, which is shown in Fig. 1.

```
Global initialization:
    $\mathcal{Q} \leftarrow \{Tcol(1), Tcol(2), \ldots, Tcol(n)\}$
    for $j = 1$ to $n$ do
        $\mathcal{S}_j \leftarrow \emptyset$
    end for

Work performed by each processor:
    while $\mathcal{Q} \neq \emptyset$ do
        pop $Tcol(j)$ from $\mathcal{Q}$ (critical section)
        let $\boldsymbol{J}$ be the supernode containing column $j$
        while column $j$ requires further cmod's from some $\boldsymbol{K} < \boldsymbol{J}$ do
            if $\mathcal{S}_j = \emptyset$ then
                wait until $\mathcal{S}_j \neq \emptyset$
            end if
            lock
            remove $\boldsymbol{K}$ from $\mathcal{S}_j$
            $q \leftarrow next(j, \boldsymbol{K})$
            if $q \leq n$ then
                $\mathcal{S}_q \leftarrow \mathcal{S}_q \cup \{\boldsymbol{K}\}$
            end if
            unlock
            $cmod(j, \boldsymbol{K})$
        end while
        $cmod(j, \boldsymbol{J})$
        $cdiv(j)$
        if $j$ is the last column of supernode $\boldsymbol{J}$ then
            $q \leftarrow next(\boldsymbol{J}, \boldsymbol{J})$
            if $q \leq n$ then
                lock
                $\mathcal{S}_q \leftarrow \mathcal{S}_q \cup \{\boldsymbol{J}\}$
                unlock
            end if
        end if
    end while
```

FIG. 1. *A parallel supernodal Cholesky factorization algorithm for shared-memory multiprocessor machines.*

The work queue is denoted by $\mathcal{Q}$ and the row structure set for updating column j is constructed in the set $\mathcal{S}_j$ as the algorithm progresses. It is worth noting that the set $\mathcal{S}_j$ contains *supernodes* that update column $L_{*,j}$, not *individual columns* that update the column. We let $next(j, \boldsymbol{K})$ be the next column after j to be updated by supernode $\boldsymbol{K}$. (Here we assume that $next(j, \boldsymbol{K})$ is $n+1$ if j is the last column updated by $\boldsymbol{K}$.) Similarly, $next(\boldsymbol{J}, \boldsymbol{J})$ is the first column to be updated by supernode $\boldsymbol{J}$.

A few comments on the algorithm in Fig. 1 are in order. First, note that a supernode $\boldsymbol{J}$ is inserted into $\mathcal{S}_q$, where $q = next(\boldsymbol{J}, \boldsymbol{J})$, only after the last column of $\boldsymbol{J}$ has been completed. We were concerned that delaying the availability of each individual column until after the supernode in which it resides is complete might hurt performance. We tried different strategies that avoided this problem, but found that the technique used in the algorithm, which is the simplest approach tried, was as effective as the others.

Second, a processor enters a "spin wait" condition whenever the set $S_j = \emptyset$ and the column modifications for $L_{*,j}$ are not complete.

Third, note that extra care is required to synchronize the *cmod*$(j, \boldsymbol{J})$ operation. Since some of the columns $j' < j$ belonging to $\boldsymbol{J}$ may not be completed, a flag has to be associated with each column to record each column's current status. The flag for a particular column is set immediately after the column has been completed. Thus one potential cause of delay in the computation is that a processor may spin while waiting for this flag to be set for a previous column in the same supernode.

Fourth, the number of `lock` and `unlock` synchronization operations required in the parallel supernodal Cholesky factorization algorithm is often much smaller than that required in the parallel column-based version. The number of synchronization operations (i.e., locking and unlocking operations) required by the latter is relatively high; it is proportional to the number of nonzeros in the Cholesky factor L. It is easy to see that the number of synchronizations required by the supernodal algorithm is proportional to the number of *compressed subscripts* [18], which is precisely the number of off-diagonal nonzero entries in the last columns of all supernodes. To illustrate the reduction in the amount of synchronization, consider a model $m \times m$ grid problem using either a five-point or a nine-point operator. Suppose the grid points are labelled using the nested dissection algorithm [9]. It is easy to show that the number of nonzeros in L is $O(m^2 \log m)$ [9] and the number of compressed subscripts is $O(m^2)$ [18]. Thus, the amount of synchronization in the parallel supernodal Cholesky factorization is reduced by a factor of $\log m$.

4. Numerical experiments.

Most of the test problems used in our numerical experiments were taken from the Harwell–Boeing sparse matrix collection [7]. In the experiments, each matrix was initially ordered using an implementation of the minimum-degree algorithm due to Liu [13], followed by a postordering of the elimination tree [15], which is helpful in computing the supernode partition [12]. Some statistics, such as the size of each matrix, nonzero counts for both A and L, number of subscripts required to represent the supernodal structure of L (denoted by $\mu(L)$), the number of supernodes in L, and the number of floating-point operations,[1] are provided in Table 1.

Throughout this section, we use `colfct` to refer to the column-based approach to Cholesky factorization, and we use `supfct` to refer to the supernode-based approach. For each approach there are two distinct but similar routines: a serial routine and a parallel routine. It is worth noting that the serial `colfct` routine is a version of SPARSPAK's `gsfct` routine that has been slightly modified for fair comparison with the other routines, which were written from scratch.

We compare parallel `colfct` and parallel `supfct` on a Cray Y-MP, a powerful vector supercomputer with 8 processors and 128 Mwords of memory. Loop unrolling to level eight was used in `supfct`, and Fortran compiler directives were used to exercise the machine's parallel capabilities and to perform the necessary synchronization operations. The code was compiled using the Fortran compiler `cf77` with optimization (the default) and preprocessing options on (i.e., `cf77 -Zu`).

The top half of Table 2 reports factorization times and speedup ratios (enclosed in parentheses) for both methods applied to some small problems in our test set. The bottom half of the table records performance data for `supfct` on the remaining problems in our test set.

[1]A single floating-point operation is either a floating-point addition, subtraction, multiplication, or division, and is denoted by "flop."

TABLE 1
Characteristics of test problems.

Problem	n	$\|A\|$	$\|L\|$	$\mu(L)$	N	flops
BCSSTK13	2,003	83,883	271,671	28,621	599	58,550,598
BCSSTK14	1,806	63,454	112,267	17,508	503	9,793,431
BCSSTK15	3,948	117,816	651,222	61,614	1,295	165,035,094
BCSSTK16	4,884	290,378	741,178	50,365	691	149,100,948
BCSSTK17	10,974	428,650	1,005,859	94,225	2,595	144,269,031
BCSSTK18	11,948	149,090	662,725	116,807	7,438	140,907,823
BCSSTK23	3,134	45,178	420,311	49,018	1,522	119,155,247
BCSSTK24	3,562	159,910	278,922	22,331	414	32,429,194
BCSSTK25	15,439	252,241	1,416,568	205,513	7,288	283,732,315
BCSSTK29	13,992	619,488	1,694,796	174,770	3,231	393,045,158
BCSSTK30	28,924	2,043,492	3,843,435	229,670	3,689	928,323,809
BCSSTK31	35,588	1,181,416	5,308,247	330,896	8,304	2,550,954,465
BCSSTK32	44,609	2,014,701	5,246,353	374,507	6,927	1,108,686,016
BCSSTK33	8,738	591,904	2,546,802	124,532	1,201	1,203,491,786
NASA1824	1,824	39,208	73,699	12,587	527	5,160,949
NASA2910	2,910	174,296	204,403	25,170	599	21,068,943
NASA4704	4,704	104,756	281,472	35,339	1,245	35,003,786
NASASRB	54,870	2,677,324	11,904,998	592,254	8,027	4,672,895,526

n: number of equations,
$|A|$: number of nonzeros in A,
$|L|$: number of nonzeros in L, including the diagonal,
$\mu(L)$: number of row subscripts required to represent the supernodal structure of L,
N: number of fundamental supernodes in L,
flops: number of floating-point operations required to compute L.

TABLE 2
Factorization times in seconds (and speedups) on a Cray Y-MP.

			Parallel				
Problem	Method	Serial	$p=1$	$p=2$	$p=4$	$p=6$	$p=8$
BCSSTK13	`colfct`	.929	1.347 (.69)	.685 (1.4)	.379 (2.5)	.314 (3.0)	.301 (3.1)
	`supfct`	.439	.493 (.89)	.249 (1.8)	.128 (3.4)	.089 (4.9)	.069 (6.4)
BCSSTK14	`colfct`	.238	.391 (.61)	.203 (1.2)	.126 (1.9)	.121 (2.0)	.121 (2.0)
	`supfct`	.156	.185 (.84)	.093 (1.7)	.048 (3.2)	.033 (4.7)	.026 (6.0)
BCSSTK15	`colfct`	2.485	3.471 (.72)	1.770 (1.4)	.962 (2.6)	.768 (3.2)	.728 (3.4)
	`supfct`	1.071	1.172 (.91)	.594 (1.8)	.299 (3.6)	.204 (5.2)	.157 (6.8)
BCSSTK16	`colfct`	2.444	3.549 (.69)	1.814 (1.3)	.995 (2.5)	.834 (2.9)	.812 (3.0)
	`supfct`	1.067	1.178 (.91)	.590 (1.8)	.299 (3.6)	.202 (5.3)	.154 (6.9)
BCSSTK17	`colfct`	2.712	4.121 (.66)	2.142 (1.3)	1.223 (2.2)	1.104 (2.5)	1.094 (2.5)
	`supfct`	1.373	1.540 (.89)	.773 (1.8)	.392 (3.5)	.267 (5.1)	.206 (6.7)
BCSSTK18	`colfct`	2.288	3.284 (.70)	1.675 (1.4)	.928 (2.5)	.767 (3.0)	.729 (3.1)
	`supfct`	1.314	1.481 (.89)	.741 (1.8)	.382 (3.4)	.265 (5.0)	.213 (6.2)
BCSSTK23	`colfct`	1.755	2.408 (.73)	1.219 (1.4)	.652 (2.7)	.514 (3.4)	.473 (3.7)
	`supfct`	.798	.879 (.91)	.441 (1.8)	.224 (3.6)	.153 (5.2)	.125 (6.4)
BCSSTK24	`colfct`	.674	1.071 (.63)	.551 (1.2)	.329 (2.0)	.302 (2.2)	.301 (2.2)
	`supfct`	.338	.381 (.89)	.190 (1.8)	.096 (3.5)	.065 (5.2)	.050 (6.8)
BCSSTK25	`supfct`	2.580	2.872 (.90)	1.433 (1.8)	.731 (3.5)	.504 (5.1)	.394 (6.5)
BCSSTK29	`supfct`	2.939	3.195 (.92)	1.602 (1.8)	.810 (3.6)	.547 (5.4)	.421 (7.0)
BCSSTK30	`supfct`	5.816	6.301 (.92)	3.154 (1.8)	1.590 (3.7)	1.073 (5.4)	.815 (7.1)
BCSSTK31	`supfct`	12.607	13.299 (.95)	6.653 (1.9)	3.330 (3.8)	2.249 (5.6)	1.706 (7.4)
BCSSTK32	`supfct`	7.773	8.491 (.92)	4.249 (1.8)	2.134 (3.6)	1.442 (5.4)	1.100 (7.1)
BCSSTK33	`supfct`	5.831	6.146 (.95)	3.074 (1.9)	1.539 (3.8)	1.040 (5.6)	.788 (7.4)
NASA1824	`supfct`	.114	.137 (.83)	.069 (1.7)	.036 (3.2)	.025 (4.6)	.019 (6.0)
NASA2910	`supfct`	.287	.331 (.87)	.166 (1.7)	.084 (3.4)	.058 (4.9)	.045 (6.4)
NASA4704	`supfct`	.429	.483 (.89)	.243 (1.8)	.124 (3.5)	.085 (5.0)	.065 (6.6)
NASASRB	`supfct`	23.563	24.850 (.95)	12.404 (1.9)	6.202 (3.8)	4.179 (5.6)	3.164 (7.4)

Not surprisingly, `supfct` performs much better than `colfct` on this machine. Loop unrolling is extremely important for good performance on the Cray Y-MP. Comparing the serial runs for the two methods, we find differences in performance ranging from a low of 53% to a high of 132%, due to loop unrolling and reduced indirect indexing in `supfct`. Similar results have been reported previously in [4]. We also find that `supfct` parallelizes much better than `colfct`. For example, on eight processors (p=8) the speedup ratios for `supfct` range from a low of 6.0 to a high of 6.9, which is quite good, especially on such small problems. The speedup ratios for `colfct`, however, are very poor, ranging from a low of 2.0 to a high of 3.7.

The performance of `supfct` on the large problems in the bottom half of the table is consistently good. Ignoring the three smallest problems in this portion of the table (NASA1824, NASA2910, and NASA4704), speedup ratios range from a low of 6.5 to a high of 7.4. On six out of seven of these problems the speedup ratio is 7.0 or greater, with the 6.5 speedup ratio reserved for the problem requiring the least work, namely BCSSTK25.

Table 3 compares the performance of `supfct` with the parallel supernodal factorization algorithm used in [19], which we will designate as `supfct-SVY`. The performance figures are expressed in Mflops,[2] as is commonly done for vector supercomputers such as the Cray Y-MP. We report the performance of both codes on those problems in our test set for which results for `supfct-SVY` were available to us. The performance data for `supfct-SVY` were obtained from an unpublished manuscript [20].

TABLE 3
Factorization computational rates (Mflops) on a Cray Y-MP.

Problem	Method	Serial	Parallel	
			p =4	p =8
BCSSTK15	`supfct`	154.1	551.9	1051.2
	`supfct-SVY`	197.8	301.1	320.8
BCSSTK16	`supfct`	139.7	498.6	968.2
	`supfct-SVY`	190.8	287.5	297.4
BCSSTK23	`supfct`	149.3	531.9	953.2
	`supfct-SVY`	191.6	293.3	315.1
BCSSTK24	`supfct`	95.9	337.8	648.5
	`supfct-SVY`	139.4	168.2	168.7
BCSSTK30	`supfct`	159.6	583.8	1139.0
	`supfct-SVY`	212.2	350.0	375.0
BCSSTK31	`supfct`	202.3	766.0	1495.3
	`supfct-SVY`	251.4	566.3	689.2
BCSSTK32	`supfct`	142.6	519.5	1007.9
	`supfct-SVY`	193.5	291.4	307.0
BCSSTK33	`supfct`	206.4	782.0	1527.3
	`supfct-SVY`	258.4	593.1	717.2
NASA1824	`supfct`	45.3	143.3	271.5
	`supfct-SVY`	64.3	69.8	69.8
NASA2920	`supfct`	73.4	250.8	468.1
	`supfct-SVY`	97.5	121.6	121.4
NASA4704	`supfct`	81.6	282.3	538.4
	`supfct-SVY`	117.0	143.5	143.5
NASASRB	`supfct`	198.3	753.4	1476.9
	`supfct-SVY`	250.6	531.6	625.2

[2]Mflops (megaflops) are millions of floating-point operations per second. Gflops (gigaflops) are billions of floating-point operations per second.

Consider the results obtained on eight processors (p=8). On six of the twelve problems, `supfct` performs the factorization at over a Gflop, with highs of 1.48 Gflops on NASASRB, 1.50 Gflops on BCSSTK31, and 1.53 Gflops on BCSSTK33. For two other problems, the computational rate is nearly a Gflop: .97 Gflop on BCSSTK16 and .95 Gflop on BCSSTK23. Thus, for 8 out of 12 of the problems, `supfct` computed the factorization at nearly a Gflop or more. Table 1 indicates that the remaining four problems (NASA1824, NASA2910, NASA4704, and BCSSTK24) are quite small. Moreover, in Table 2 we see that serial `supfct` requires less than half a second to factor any of these four matrices.

The performance of `supfct-SVY` is much poorer due to the problems with this approach mentioned earlier in §2. The code runs at less than a Gflop on every problem, despite having significantly higher serial efficiency due to assembly language programming of the compute-intensive kernel routines and other machine-specific optimizations. Our parallel implementation of `supfct` is a Fortran 77 code, with no machine-specific optimizations.

5. Concluding remarks. We have implemented a simple new parallel sparse Cholesky factorization algorithm for shared-memory multiprocessors. Our left-looking algorithm uses techniques from [4] and [10]: it uses supernodes to reduce indirect addressing and memory traffic [4], and it decomposes the computation into column tasks $Tcol(j)$ and schedules these tasks dynamically on the available processors [10]. Incorporation of supernodes into the algorithm in [10] reduces the synchronization overhead required to manage the row structure sets $\mathcal{S}_q$ from $O(|L|)$ to $O(\mu(L))$. In practice, $\mu(L)$ is often much smaller than $|L|$; consequently, contention for the critical sections is likely to be higher in `colfct` than in `supfct`, particularly on machines such as the Cray Y-MP, which have a limited number of lock variables.

Acknowledgment. The authors would like to thank Dr. Phuong Vu and Cray Research Inc. for providing us with dedicated time on a Cray Y-MP.

REFERENCES

[1] P. Amestoy and I. Duff, *Vectorization of a multiprocessor multifrontal code*, Internat. J. Supercomput. Appl., 3 (1989), pp. 41–59.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK: A portable linear algebra library for high-performance computers*, in Proceedings of Supercomputing '90, IEEE Press, Washington, DC, 1990, pp. 1–10.

[3] C. Ashcraft, *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Tech. Rep. ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, WA, 1987.

[4] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Internat. J. Supercomput. Appl., 1 (1987), pp. 10–30.

[5] E. Chu, A. George, J. W.-H. Liu, and E. G.-Y. Ng, *User's guide for SPARSPAK-A: Waterloo sparse linear equations package*, Tech. Rep. CS-84-36, Univ. of Waterloo, Waterloo, Ontario, Canada, 1984.

[6] J. Dongarra, I. Duff, J. D. Croz, and S. Hammarling, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[7] I. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[8] I. Duff and J. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[9] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[10] A. GEORGE, M. HEATH, J. W.-H. LIU, AND E. G.-Y. NG, *Solution of sparse positive definite systems on a shared memory multiprocessor*, Internat. J. Parallel Programming, 15 (1986), pp. 309–325.

[11] A. GEORGE AND J. W.-H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[12] J. LIU, E. NG, AND B. PEYTON, *On finding supernodes for sparse matrix computations*, Tech. Rep. ORNL/TM-11563, Oak Ridge National Laboratory, Oak Ridge, TN, 1990. To appear in SIAM J. Matrix Anal. Appl.

[13] J. W.-H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[14] ———, *Computational models and task scheduling for parallel sparse Cholesky factorization*, Parallel Comput., 3 (1986), pp. 327–342.

[15] ———, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

[16] A. POTHEM, *Simplicial cliques, shortest elimination trees, and supernodes in sparse Cholesky factorization*, Tech. Rep. CS-88-13, Dept. of Computer Science, The Pennsylvania State Univ., University Park, PA, 1988.

[17] E. ROTHBERG AND A. GUPTA, *Efficient sparse matrix factorization on high-performance workstations—exploiting the memory hierarchy*, ACM Trans. Math. Software, 17 (1991), pp. 313–334.

[18] A. SHERMAN, *On the efficient solution of sparse systems of linear and nonlinear equations*, Ph.D. thesis, Yale University, New Haven, CT, 1975.

[19] H. SIMON, P. VU, AND C. YANG, *Sparse matrix at* 1.68 *Gflops*, Tech. Report, Boeing Computer Services, Seattle, WA, 1989.

[20] C. YANG, *A vector/parallel implementation of the multifrontal method for sparse symmetric definite linear systems on the Cray Y-MP*, Cray Research Inc., Mendota Heights, MN, 1990.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 770–782, July 1993

GENERATING LINEAR AND LINEAR-QUADRATIC BILEVEL PROGRAMMING PROBLEMS*

PAUL H. CALAMAI[†] AND LUIS N. VICENTE[‡]

Abstract. This paper describes a technique for generating both linear and linear-quadratic bilevel programming problems. The method is based on combining a number of single-parameter two-variable problems to obtain a separable multivariable problem with a number of desirable properties. The separability is then disguised using a simple transformation.

The proposed technique, which requires very little computational effort, allows the user control over, among other things, the number and type of minima and the data density.

Key words. bilevel programming, test problems, separable programs

AMS subject classifications. 90C30, 90D05

1. Introduction. An important field of mathematical programming is bilevel programming [5], [9], [11]. One reason for this interest is that problems involving hierarchical decision making can be modeled as bilevel problems [1], [2], [6]. Although several techniques have been designed for solving linear and linear-quadratic bilevel programming problems [3], [7], [8], [12], little has been done to compare the computational virtues of these techniques. One likely reason for this omission is the lack of an acceptably large and diverse set of test problems. In this paper we describe a computationally simple procedure for constructing problems that can be used to this end. Much of this work is based on a related technique for generating quadratic bilevel programming problems [4].

The method described here gives the user control over a number of important problem features that can be adjusted to test the performance of the various solution techniques. We demonstrate how the number and type of minima can be controlled and how the data density can be adjusted. Since several solution techniques start by solving the relaxed linear program that corresponds to the bilevel problem, we also show how the diversity between the solutions of these two problems can be controlled.

In §2 we describe how the linear bilevel programming problems are constructed. We start by introducing the single-parameter, two-variable, bilevel programming problem that is our fundamental building block. The solution properties of these problems are described using simple geometry. We then demonstrate how these simple problems can be combined to produce a separable linear bilevel problem with chosen properties. In §3 we describe how our basic linear bilevel problems can be modified and extended. We show how the number of inequality constraints can be reduced and how equality constraints can be added. The modifications required for constructing linear-quadratic bilevel programming problems are also included in this section. The transformation that is used to disguise the separability of the constructed problems is then described in §4. In §5 we illustrate our technique with an example. The paper concludes with §6, where

*Received by the editors December 4, 1991; accepted for publication (in revised form) June 24, 1992. Support of this work has been provided by Instituto Nacional de Investigação Científica de Portugal (INIC) contract 89/EXA/5 and by Natural Sciences and Engineering Research Council of Canada operating grant 5671.

[†]Department of Systems Design Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. This paper was completed while this author was on a research sabbatical at the Universidade de Coimbra, Portugal.

[‡]Departamento de Matemática, Universidade de Coimbra, 3000 Coimbra, Portugal.

we report conclusions and demonstrate that the proposed method generates problems that meet the requirements of three different solution techniques.

2. The linear problem. Our intention is to construct purely linear bilevel programming (LBP) problems with a number of favorable properties. We accomplish this by starting with a simple single-parameter problem in one upper-level and one lower-level variable. We then combine several of these problems to obtain a separable multiparameter multivariable linear bilevel programming problem with several desirable properties. The separability of this problem is then disguised, without destroying these properties, using a simple, nonsingular transformation of variables.

2.1. A one-parameter, two-variable LBP. Consider the following linear bilevel programming problem in the variables x_k and y_k, denoted $LBP(\rho_k)$:

$$\begin{aligned}&\min_{x_k,y_k} L_k(x_k,y_k) = (3-x_k)+y_k,\\&\text{subject to } (x_k,y_k)\in\Omega_U^k\end{aligned}$$

(the upper-level problem), where $y_k = y(x_k)$ solves (the lower-level problem)

$$\begin{aligned}&\min_{y_k} l_k(x_k,y_k) = -y_k,\\&\text{subject to } (x_k,y_k)\in\Omega_L(\rho_k),\end{aligned}$$

with $3 \le \rho_k \le 9$, $\Omega_U^k = \{x_k : 1 \le x_k \le 3\}$, and $\Omega_L(\rho_k)$ consisting of those points (x_k, y_k) satisfying

$$\begin{aligned}x_k + y_k &\le \rho_k,\\-2x_k + y_k &\le 0.\end{aligned}$$

The relaxed feasible region for $LBP(\rho_k)$, $\Omega(\rho_k) = \Omega_U^k \bigcap \Omega_L(\rho_k)$, is thus the (unbounded) region bounded on the left by $x_k \ge 1$, on the right by $x_k \le 3$, and above by $-2x_k + y_k \le 0$ (to the left) and $x_k + y_k \le \rho_k$ (to the right).

The set of all feasible points of problem $LBP(\rho_k)$ is called the *induced region* (see [12, pp. 9–10] for a complete mathematical description).

PROPOSITION 2.1. *For problem $LBP(\rho_k)$ the induced region, denoted S, consists of the union of the following two sets:*

$$\begin{aligned}S_1 &= \{(x_k,y_k)\in\Omega(\rho_k) : -2x_k+y_k=0\},\\S_2 &= \{(x_k,y_k)\in\Omega(\rho_k) : x_k+y_k=\rho_k\},\end{aligned}$$

which describe two line segments in $\Omega(\rho_k)$.

Proof. The proof follows directly upon applying the Karush–Kuhn–Tucker conditions to the corresponding lower-level problem. □

The solution of problem $LBP(\rho_k)$ occurs at some point $(x_k, y(x_k)) \in S$ which depends on the value of the parameter ρ_k. There are five cases to consider:

1. Case 1 (Fig. 1), where $\rho_k = 3$.
2. Case 2 (Fig. 2), where $\rho_k = 7$.
3. Case 3 (Fig. 3), where $\rho_k = 9$.
4. Case 4, where $3 < \rho_k < 7$.
5. Case 5, where $7 < \rho_k < 9$.

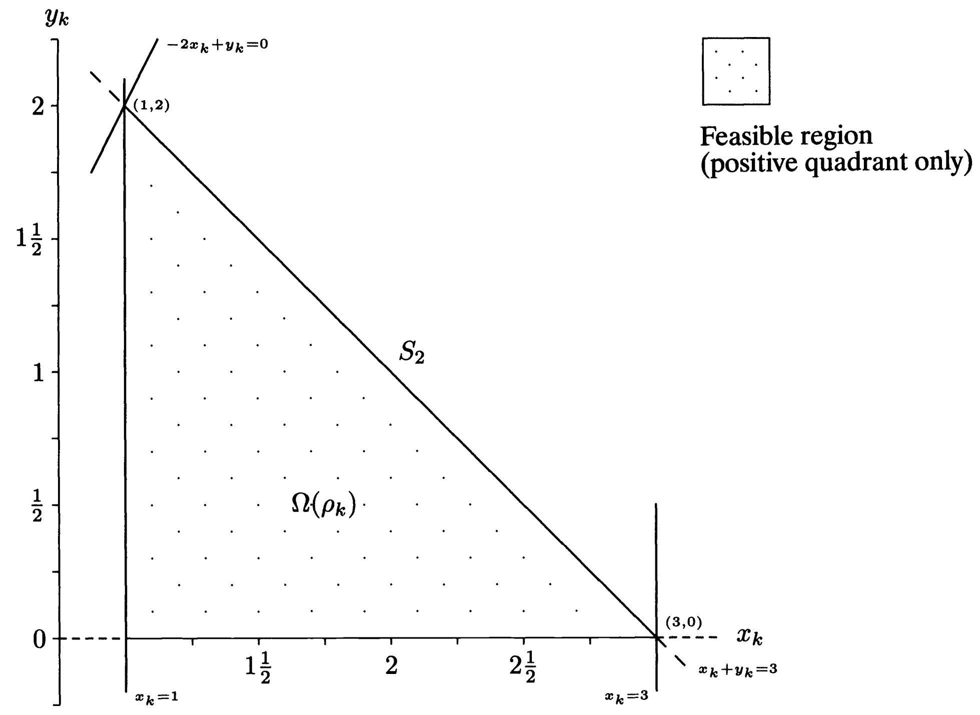


FIG. 1. *Case* 1, *where* $\rho_k = 3$.

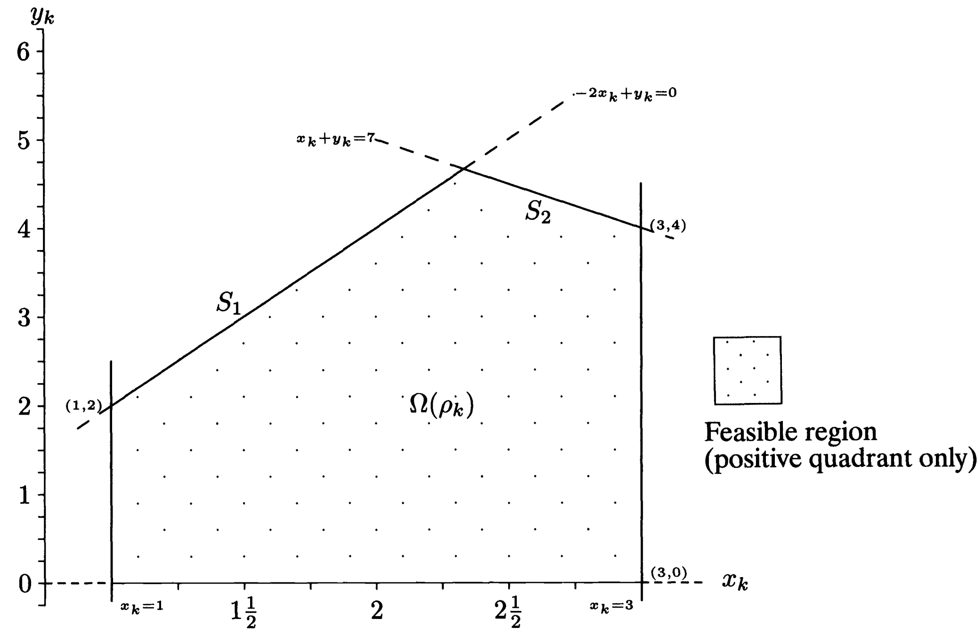


FIG. 2. *Case* 2, *where* $\rho_k = 7$.

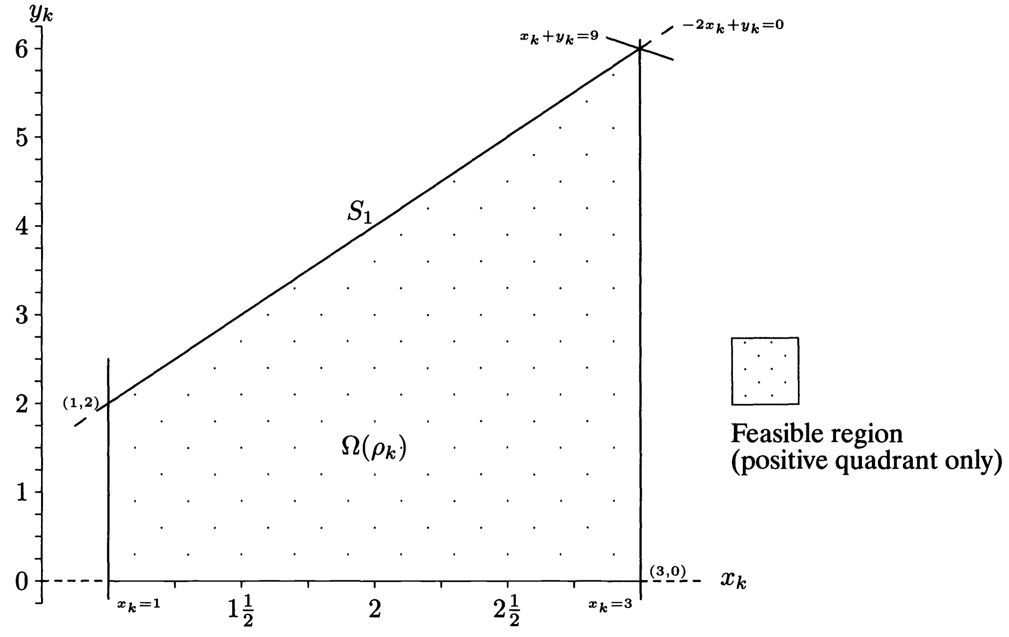


FIG. 3. *Case* 3, *where* $\rho_k = 9$.

What follows is an examination of problem $LBP(\rho_k)$ for each of these five cases. In each instance, the solutions are determined geometrically. This is a simple matter since, for $(x_k, y_k) \in S$, we have

$$\begin{aligned}\min_{x_k,y_k} L_k(x_k,y_k) &= (3-x_k)+y_k \\ &= |x_k-3|+|y_k| \\ &= \left\| \begin{pmatrix} x_k \\ y_k \end{pmatrix} - \begin{pmatrix} 3 \\ 0 \end{pmatrix} \right\|_1 \end{aligned}$$

This means that the global minima of problem $LBP(\rho_k)$, denoted (x_k^G, y_k^G), are simply those points in S closest to the point $(3,0)$ in the l_1 sense. Similarly, the local minima of problem $LBP(\rho_k)$, denoted (x_k^L, y_k^L), are points in S that are closer (in the l_1 sense) to the point $(3,0)$ than points in S in some nonempty neighborhood of (x_k^L, y_k^L).

Case 1. $\rho_k = 3$. In this case, depicted in Fig. 1, the set S equals S_2, and therefore includes the point $(3,0)$. Thus $(x_k^G, y_k^G) = (3,0)$ with $L_k(x_k^G, y_k^G) = 0$.

Case 2. $\rho_k = 7$. In this case, depicted in Fig. 2, the set S is the union of S_1 and S_2. By observing the l_1 contours centered at the point $(3,0)$, we find that $LBP(\rho_k)$ has two (strict) global minima, namely $(x_k^G, y_k^G) = (1,2)$ and $(x_k^G, y_k^G) = (3,4)$, with $L_k(x_k^G, y_k^G) = 4$.

Case 3. $\rho_k = 9$. Figure 3 depicts this case, where the set S equals S_1. In this case the closest point in S, in the l_1 sense, to $(3,0)$ is the point $(1,2)$. Thus $(x_k^G, y_k^G) = (1,2)$ with $L_k(x_k^G, y_k^G) = 4$.

Case 4. $3 < \rho_k < 7$. This case is similar to the situation depicted in Fig. 2 except that the line $x_k + y_k = 7$ is replaced with the line $x_k + y_k = \rho_k$ (i.e., the line $x_k + y_k = 7$ is shifted *down* $7 - \rho_k$ units), subsequently changing S_1 and S_2 (and therefore S). The closest point in S, in the l_1 sense, to $(3, 0)$ occurs at the intersection of S_2 with the line $x_k = 3$. Thus $(x_k^G, y_k^G) = (3, \rho_k - 3)$ with $L_k(x_k^G, y_k^G) = \rho_k - 3$.

In addition, the point $(1, 2)$, the left endpoint of S_1, is closer, in the l_1 sense, to the point $(3, 0)$ than any other point in S_1. Thus $(x_k^L, y_k^L) = (1, 2)$ with $L_k(x_k^L, y_k^L) = 4$.

Case 5. $7 < \rho_k < 9$. This case is also similar to the situation depicted in Fig. 2, except that this time the line $x_k + y_k = 7$ is shifted *up* $\rho_k - 7$ units (i.e., the line $x_k + y_k = 7$ is replaced with the line $x_k + y_k = \rho_k$), subsequently changing S_1 and S_2 (and therefore S). The closest point in S, in the l_1 sense, to $(3, 0)$ for this case occurs at the intersection of S_1 with the line $x_k = 1$ (as in Case 3). Thus $(x_k^G, y_k^G) = (1, 2)$ with $L_k(x_k^G, y_k^G) = 4$.

The point $(3, \rho_k - 3)$, the right endpoint of S_2, is closer, in the l_1-sense, to the point $(3, 0)$ than any other point in S_2. Thus $(x_k^L, y_k^L) = (3, \rho_k - 3)$ with $L_k(x_k^L, y_k^L) = \rho_k - 3$.

In the next section we demonstrate how a (separable) multiparameter multivariable linear bilevel programming problem, with specific properties, can be constructed by combining these simple one-parameter, two-variable problems.

2.2. A simple, separable parametric LBP. Let nx and ny be the desired number of upper-level and lower-level variables of the linear bilevel problem, respectively. In addition, let $m = \min\{nx, ny\}$.

Consider the following problem, which we denote $LBP(\rho)$:

$$\begin{aligned}\min_{x,y} L(x, y) &= \sum_{k=1}^{m} L_k(x_k, y_k) + \sum_{m<k\le nx} (3 - x_k) + \sum_{m<k\le ny} y_k \\ &= \sum_{k=1}^{nx} (3 - x_k) + \sum_{k=1}^{ny} y_k,\end{aligned}$$

subject to $(x, y) \in \Omega_U$

(the upper-level problem), where $y = y(x)$ solves (the lower-level problem)

$$\min_{y} l(x, y) = \sum_{k=1}^{m} l_k(x_k, y_k) = -\sum_{k=1}^{m} y_k,$$

subject to $(x, y) \in \Omega_L(\rho)$

with $3 \le \rho_k \le 9$, for $k = 1, \ldots, m$,

$$\Omega_U = \bigcap_{k=1}^{m} \Omega_U^k \quad \bigcap_{m<k\le nx} \{x_k : x_k \le 3\},$$

and

$$\Omega_L(\rho) = \bigcap_{k=1}^{m} \Omega_L(\rho_k) \quad \bigcap_{m<k\le ny} \{y_k : y_k \ge 0\}.$$

In addition, define the corresponding *relaxed* linear program $LP(\rho)$ as

$$\min_{x,y} \{L(x, y) : (x, y) \in \Omega_U \cap \Omega_L(\rho)\}.$$

From this construction we can easily see that the solution of the multiparameter problem $LBP(\rho)$ is simply composed of the solutions of the m single-parameter problems $LBP(\rho_k)$, $k = 1, \ldots, m$, plus the solution of the subproblem

$$\min \sum_{m<k\le nx} (3 - x_k) + \sum_{m<k\le ny} y_k,$$

subject to

$$x_k \leq 3, \qquad m < k \leq nx,$$
$$y_k \geq 0, \qquad m < k \leq ny.$$

If we define the five sets M_1 through M_5 (with corresponding cardinalities m_1 through m_5) as

$$\begin{aligned} M_1 &= \{k \in M : \rho_k = 3\}, \\ M_2 &= \{k \in M : \rho_k = 7\}, \\ M_3 &= \{k \in M : \rho_k = 9\}, \\ M_4 &= \{k \in M : 3 < \rho_k < 7\}, \\ M_5 &= \{k \in M : 7 < \rho_k < 9\}, \end{aligned}$$

where $M = \{1, \ldots, m\}$, then the following properties hold.

PROPERTY 1. Problem $LBP(\rho)$ has 2^{m_2} global minima with $x_k^G = 3$ $(m < k \leq nx)$, $y_k^G = 0$ $(m < k \leq ny)$, and

$$(x_k^G, y_k^G) = \begin{cases} (3,0), & k \in M_1, \\ (1,2) \text{ or } (3,4), & k \in M_2, \\ (1,2), & k \in M_3 \cup M_5, \\ (3, \rho_k - 3), & k \in M_4, \end{cases}$$

yielding $L(x^G, y^G) = 4(m_2 + m_3 + m_5) + \sum_{k \in M_4} (\rho_k - 3)$.

This property follows directly from the manner in which problem $LBP(\rho)$ is constructed and from the previous geometric analysis of problems $LBP(\rho_k)$, $k \in M$.

PROPERTY 2. Problem $LBP(\rho)$ has local minima (that are not global) only when $m_4 + m_5 > 0$. In this case there are $2^{m_2+m_4+m_5} - 2^{m_2}$ such minima. These local minima occur whenever one or more of the following hold

$$(x_k, y_k) = \begin{cases} (1,2), & k \in M_4, \\ (3, \rho_k - 3), & k \in M_5, \end{cases}$$

with all remaining components defined as in Property 1.

This property, like Property 1, follows directly from the manner in which problem $LBP(\rho)$ is constructed and from the geometric properties of problems $LBP(\rho_k)$, $k \in M$.

PROPERTY 3. The global minima of problem $LBP(\rho)$ do not coincide with any solution of the corresponding relaxed linear program $LP(\rho)$.

This result follows trivially since problem $LP(\rho)$ is unbounded. In fact, this result still holds (as long as $m_1 < m$) if one adds what might be considered the natural lower-bound constraints $y_k \geq 0$, $k = 1, \ldots, ny$, since the solution to the corresponding relaxed LP yields $x_k = 3$, $k = 1, \ldots, nx$, and $y_k = 0$, $k = 1, \ldots, ny$, which differs from the minima of $LBP(\rho)$ in $2(m - m_1)$ variables.

PROPERTY 4. The gradients, in the lower-level variables y, of the active constraints at the minima of problem $LBP(\rho)$ are linearly independent, and the complementarity conditions associated with the corresponding lower-level problem are *strictly* satisfied at these minima.

This property is a direct consequence of the separability of problem $LBP(\rho)$ and the strict complementarity of the minima of problems $LBP(\rho_k)$, $k = 1, \ldots, m$.

3. Extensions and modifications. Problem $LBP(\rho)$ can be extended and modified in a number of ways. In the subsections that follow we show how the number of inequality constraints in problem $LBP(\rho)$ can be reduced, how equality constraints can be added to problem $LBP(\rho)$, and how our technique can be modified to produce bilevel problems with quadratic lower-level objectives (i.e., linear-quadratic bilevel programs).

3.1. Reducing the number of inequality constraints. When $k \in M_1$, the constraints

$$x_k \geq 1 \quad \text{and} \quad -2x_k + y_k \leq 0$$

are redundant in problems $LBP(\rho_k)$. Similarly, when $k \in M_3$, the constraints

$$x_k \leq 3 \quad \text{and} \quad x_k + y_k \leq 9$$

are redundant in problems $LBP(\rho_k)$. Thus $2(m_1+m_3)$ constraints can be removed from problem $LBP(\rho)$ without consequence.

An additional $2(m-\bar{m})$ constraints can be removed by setting $\bar{m} < m = \min\{nx, ny\}$ and replacing the constraints

$$\begin{aligned} -x_k &\leq -1, & k &= \bar{m}+1, \dots, m, \\ x_k + y_k &\leq \rho_k, & k &= \bar{m}+1, \dots, m, \\ -2x_k + y_k &\leq 0, & k &= \bar{m}+1, \dots, m, \end{aligned}$$

with the constraints

$$y_k \geq 0, \qquad k = \bar{m}+1, \dots, m.$$

This requires redefining M_1 through M_5 (and their corresponding cardinalities m_1 through m_5) after setting $M = \{1, \dots, \bar{m}\}$.

With this modification, the minima of $LBP(\rho)$ satisfy $(x_k^G, y_k^G) = (3, 0)$ when $k \in \{\bar{m}+1, \dots, m\}$.

3.2. Adding equality constraints. Equality constraints can be added to problem $LBP(\rho)$ in a number of ways without changing its minima. Those analogous to the ones presented in §6.2 of [4] are:

1. $x_i - x_j = 0$, where $i \neq j$ and $i, j \in \{m+1, \dots, nx\}$ when $nx > ny = m$.
2. $y_i - y_j = 0$, where $i \neq j$ and $i, j \in \{m+1, \dots, ny\}$ when $ny > nx = m$.
3. $x_i - y_j = 3$, where $i, j \in M_1$ when $m_1 > 0$.
4. $x_i - y_j = 3$, where $i, j \in \{\bar{m}+1, \dots, m\}$, and $\bar{m}$ and M_1 through M_5 are defined as in the previous section.

In order to ensure the linear independence of the gradients (with respect to the lower-level variables y) of the constraints at optimality, the same conditions as those stated in §6.2 of [4] must hold.

3.3. Replacing the lower-level objective with a quadratic. After redefining $\bar{m}$ and M_1 through M_5 as in §3.1, and removing the constraints

$$-2x_k + y_k \leq 0, \qquad k = 1, \dots, \bar{m},$$

the linear, lower-level objective $l(x, y)$ can be replaced with the quadratic

$$\min_y q(x, y) = \frac{1}{2} \sum_{k=1}^{\bar{m}} y_k (y_k - 4x_k) - \sum_{k=\bar{m}+1}^{m} y_k$$

without affecting the foregoing analysis (since the induced region S is the same for (x_k, y_k) when $k \in M$).

With this modification we obtain a linear-quadratic bilevel programming problem, which we denote $LQBP(\rho)$.

4. The transformation. In order to describe our technique for disguising the separability of problem $LBP(\rho)$, we define problem $LBP(c, s, A_x, A_y, b)$ as

$$\min_{x,y} L(x,y) = c_x^T x + c_y^T y + \kappa$$

(the upper-level problem), where $y = y(x)$ solves (the lower-level problem)

$$\begin{aligned}&\min_{y} l(x,y) = s_x^T x + s_y^T y,\\ &\text{subject to } A_x x + A_y y \le b,\end{aligned}$$

with

$$\begin{gathered} c = \begin{bmatrix} c_x \\ c_y \end{bmatrix}, \qquad s = \begin{bmatrix} s_x \\ s_y \end{bmatrix}, \\ c_x, s_x, x \in \boldsymbol{R}^{nx}, \qquad c_y, s_y, y \in \boldsymbol{R}^{ny}, \\ A_x \in \boldsymbol{R}^{\beta \times nx}, \quad A_y \in \boldsymbol{R}^{\beta \times ny}, \quad b \in \boldsymbol{R}^{\beta} \quad \text{and} \quad \kappa \in \boldsymbol{R}. \end{gathered}$$

With the following substitutions,

$$c_x = -1_{nx}, \quad c_y = 1_{ny}, \quad s_x = 0_{nx}, \quad (s_y)_i = \begin{cases} -1, & 1, \ldots, m, \\ 0, & \text{otherwise,} \end{cases} \quad \kappa = 3nx,$$

$$A_x = \begin{bmatrix} P_x \\ -2P_x \\ \frac{1}{3}P_x \\ -P_x \\ Q_x \end{bmatrix}, \quad A_y = \begin{bmatrix} P_y \\ P_y \\ \mathbf{0} \\ \mathbf{0} \\ Q_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} \rho \\ 0_m \\ 1_m \\ -1_m \\ \epsilon \end{bmatrix},$$

where

$\mathbf{0}$ is the $m \times ny$ zero matrix and $\rho^T = [\rho_1 \cdots \rho_m]$,

1_γ is a ones-vector of length γ and 0_γ is a zeros-vector of length γ,

$$P_x \in \boldsymbol{R}^{m \times nx} \text{ and } P_y \in \boldsymbol{R}^{m \times ny} \text{ satisfy } P_{ij} = \begin{cases} 1, & 1 \le i = j \le m, \\ 0, & \text{otherwise,} \end{cases}$$

and, for $\alpha = \max\{nx, ny\} - m$,

$$Q_x \in \boldsymbol{R}^{\alpha \times nx} \text{ satisfies } (Q_x)_{ij} = \begin{cases} 0, & ny > nx \quad \text{or} \quad j \ne m + i, \\ \frac{1}{3}, & \text{otherwise,} \end{cases}$$

$$Q_y \in \boldsymbol{R}^{\alpha \times ny} \text{ satisfies } (Q_y)_{ij} = \begin{cases} 0, & nx > ny \quad \text{or} \quad j \neq m+i, \\ -1, & \text{otherwise,} \end{cases} \qquad \text{and}$$

$$\epsilon \in \boldsymbol{R}^{\alpha} \text{ satisfies } \epsilon_i = \begin{cases} 1, & nx > ny, \\ 0, & ny > nx, \end{cases}$$

problem $LBP(c, s, A_x, A_y, b)$ is equivalent to problem $LBP(\rho)$.

Now, for $n = nx + ny$, define the order-n matrix $\mathcal{M} = HDH$, where

$$H = \begin{bmatrix} H_x & 0 \\ 0 & H_y \end{bmatrix}$$

is a block-diagonal matrix with H_x and H_y constructed as random Householder matrices using

$$H_x = I_{nx} - 2v_x v_x^T, \quad \text{with } v_x^T v_x = 1 \quad \text{and} \quad v_x \in \boldsymbol{R}^{nx} \text{ sparse,}$$

$$H_y = I_{ny} - 2v_y v_y^T, \quad \text{with } v_y^T v_y = 1 \quad \text{and} \quad v_y \in \boldsymbol{R}^{ny} \text{ sparse,}$$

and D is a positive definite diagonal matrix with two-norm condition number $\kappa_2(D) = 10^6$, and let $\mathcal{W} = \mathcal{M}^{-1} = HD^{-1}H$ and $A = \begin{bmatrix} A_x & A_y \end{bmatrix}$.

PROPOSITION 4.1. *Problem $LBP(c, s, A_x, A_y, b)$ (and problem $LBP(\rho)$) in the variables x and y is equivalent to problem* LBP$(\mathcal{M}c, \mathcal{M}s, A\mathcal{M}, b)$ *in the variables $\bar{x}$ and $\bar{y}$ under the nonsingular transformation*

$$\begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} = \mathcal{W} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Proof. For

$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathcal{M} \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix},$$

problem $LBP(c, s, A_x, A_y, b)$ becomes

$$\min_{\bar{x},\bar{y}} L(\bar{x}, \bar{y}) = \begin{bmatrix} c_x \\ c_y \end{bmatrix}^T \mathcal{M} \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} + \kappa$$

subject to $\bar{y} = \bar{y}(\bar{x})$ solving

$$\min_{\bar{y}} l(\bar{y}) = \begin{bmatrix} s_x \\ s_y \end{bmatrix}^T \mathcal{M} \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix},$$

$$\text{subject to } [A\mathcal{M}] \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} \leq b,$$

which is problem $LBP(\mathcal{M}c, \mathcal{M}s, A\mathcal{M}, b)$ in the variables $\bar{x}$ and $\bar{y}$. □

Thus,

$$\begin{bmatrix} \bar{x}^G \\ \bar{y}^G \end{bmatrix} = \mathcal{W} \begin{bmatrix} x^G \\ y^G \end{bmatrix}$$

is a global solution to the transformed problem and this one-to-one correspondence holds for all minima of problem LBP.

Notes.

1. A similar technique can be applied to problem $LQBP(\rho)$ to disguise its separability.

2. The sparsity of v_x and v_y controls the sparsity of $\mathcal{M}$ (and consequently, the sparsity of the data).

3. The spectrum of D controls the spectrum of $\mathcal{M}$ (and consequently, affects the geometry of the problem).

4. The lower-level objective of problem $LQBP(\rho)$ is strictly convex in y.

5. The upper-level constraints of problem $LBP(\rho)$ have been moved into the lower level of problem $LBP(c, s, A_x, A_y, b)$ in order to simplify notation.

6. A paper of Marcotte and Savard [10] was brought to our attention after completing this work. In their report a transformation similar to that used here is used to establish the equivalence between bilevel problems and to prove that solutions to bilevel problems are not necessarily Pareto optimal.

5. Example. In order to illustrate some of the ideas that have been presented, consider the (untransformed) linear bilevel programming problem obtained when the following values are used for the control parameters:

$$nx = 4, \quad ny = 2 \quad \text{and} \quad m = 2,$$
$$m_1 = m_2 = m_3 = 0 \quad \text{and} \quad m_4 = m_5 = 1,$$

and

$$\rho_1 = 5 \quad \text{and} \quad \rho_2 = 8.$$

This corresponds to the following bilevel problem:

$$\min_{x,y} L(x,y) = \sum_{i=1}^{2} ((3 - x_i) + y_i) + \sum_{i=3}^{4} (3 - x_i),$$

subject to $y = y(x)$ solving

$$\min_{y} l(x,y) = -\sum_{i=1}^{2} y_i,$$

subject to

$$\begin{array}{rrrrrrr}
x_1 & & & & +\, y_1 & & \le\ 5, \\
 & x_2 & & & & +\, y_2 & \le\ 8, \\
-2x_1 & & & & +\, y_1 & & \le\ 0, \\
 & -\ 2x_2 & & & & +\, y_2 & \le\ 0, \\
x_1 & & & & & & \le\ 3, \\
 & x_2 & & & & & \le\ 3, \\
-x_1 & & & & & & \le -1, \\
 & -\ x_2 & & & & & \le -1, \\
 & & x_3 & & & & \le\ 3, \\
 & & & x_4 & & & \le\ 3,
\end{array}$$

where

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \in \boldsymbol{R}^4 \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \in \boldsymbol{R}^2.$$

Now suppose the following data was used in the transformation:

$$v_x^T = \begin{bmatrix} -0.7 & 0.7 & 0.1 & -0.1 \end{bmatrix}, \qquad v_y^T = \begin{bmatrix} 0.8 & -0.6 \end{bmatrix},$$

and

$$D = \text{diag}(20, 10, 20, 10, 10, 20).$$

This would yield the following (transformed) linear bilevel programming problem:

$$\min_{x,y} L(x,y) = \begin{bmatrix} -11.6 \\ -18.4 \\ -21.2 \\ -8.8 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 21.904 \\ 13.472 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + 12$$

(the upper-level problem) subject to $y = y(x)$ solving (the lower-level problem)

$$\min_{y} l(x,y) = \begin{bmatrix} -21.904 \\ -13.472 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

subject to

$$\begin{array}{rrrrrrl}
10.2x_1 & & +\ 1.4x_3 & & +\ 19.216y_1 & +\ \ 2.688y_2 & \le\ \ 5,\\
 & 19.8x_2 & & -\ \ 1.4x_4 & +\ \ 2.688y_1 & +\ 10.784y_2 & \le\ \ 8,\\
-20.4x_1 & & -\ \ 2.8x_3 & & +\ 19.216y_1 & +\ \ 2.688y_2 & \le\ \ 0,\\
 & -39.6x_2 & & +\ \ 2.8x_4 & +\ \ 2.688y_1 & +\ 10.784y_2 & \le\ \ 0,\\
10.2x_1 & & +\ 1.4x_3 & & & & \le\ \ 3,\\
 & 19.8x_2 & & -\ \ 1.4x_4 & & & \le\ \ 3,\\
-10.2x_1 & & -\ \ 1.4x_3 & & & & \le -1,\\
 & -19.8x_2 & & +\ \ 1.4x_4 & & & \le -1,\\
1.4x_1 & & +\ 19.8x_3 & & & & \le\ \ 3,\\
 & -1.4x_2 & & +\ 10.2x_4 & & & \le\ \ 3,
\end{array}$$

where

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \in \boldsymbol{R}^4 \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \in \boldsymbol{R}^2.$$

Since $m_2 = 0$ and $m_4 + m_5 = 2$ this problem has $2^{m_2+m_4+m_5} - 2^{m_2} = 3$ local solutions and a unique global minimum with value $L_G = 4(m_2+m_3+m_5)+(\rho_1-3) = 6$.

6. Remarks and conclusions. In designing these test problems careful attention has been placed on conforming to the requirements of several solution techniques. We elaborate on this point by considering three available solution techniques.

SLCP approaches [7], [8]. The requirements of these sequential linear complementarity approaches are met since the lower-level problems can be replaced with the corresponding Karush–Kuhn–Tucker conditions. In addition, for the linear-quadratic bilevel problems an initial basis for the first LCP is readily available.

Exact penalty approaches [3]. For the pure linear bilevel problems generated using the proposed techniques

• the feasible region Ω is nonempty,

• the lower-level variables satisfy a strict complementarity condition at all local minima, and

• the lower-level objective function is bounded below on the feasible region and the upper-level objective function is bounded below in the induced region.

Thus, the requirements of this exact penalty function approach are met.

The requirements of these techniques also hold for the linear-quadratic case, as demonstrated in [4].

Branch and bound techniques [12, pp. 19–22]. In respect to the pure linear bilevel problems, the requirements for these techniques are met since there is always an optimal solution for the generated problems and since the lower-level problem has a unique solution in y for every fixed x.

We have described a method for generating both linear and linear-quadratic bilevel test problems. The method requires very little computational effort (i.e., no subproblems need be solved) and gives the user control over the number and type of minima, the sparsity of the data, the diversity from the corresponding relaxed problem and, to a lesser extent, the geometry of the problems generated.

Note. A Fortran 77 code that implements the technique described in this paper can be obtained by sending an e-mail request to phcalamai@dial.waterloo.edu.

REFERENCES

[1] J. Bard, *Coordination of a multidivisional organization through two levels of management*, Omega, 11 (1983), pp. 457–468.

[2] O. Ben-Ayed, C. Blair, and D. Boyce, *Construction of a real world bilevel linear program of the highway design problem*, Faculty Paper No. 1464, College of Commerce and Business Administration, Univ. of Illinois at Urbana-Champaign, 1988.

[3] Z. Bi, P. Calamai, and A. Conn, *An exact penalty function approach for the linear bilevel programming problem*, Tech. Rep. #167-O-310789, Dept. of Systems Design Engineering, Univ. of Waterloo, Ontario, Canada, 1989.

[4] P. Calamai and N. Vicente, *Generating quadratic bilevel programming problems*, ACM Trans. Math. Software, to appear.

[5] Y. Dirickx and L. Jennegren, *Systems Analysis by Multilevel Methods: With Applications to Economics and Management*, John Wiley, New York, 1979.

[6] 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.

[7] J. Júdice and A. Faustino, *The linear-quadratic bilevel programming problem*, Infor, to appear.

[8] ———, *A sequential LCP method for bilevel linear programming*, Ann. Oper. Res., to appear.

[9] C. Kolstad, *A review of the literature on bi-level mathematical programming*, Los Alamos Tech. Rep. LA-10284-MS, UC-32, Los Alamos National Laboratory, Los Alamos, NM, 1985.

[10] P. Marcotte and G. Savard, *A note on the Pareto optimality of solutions to the linear bilevel programming problem*, Comput. Oper. Res., 18 (1991), pp. 355–359.

[11] M. Mesanovic, D. Macko, and Y. Takahara, *Theory of Hierarchical, Multilevel Systems*, Academic Press, New York and London, 1970.

[12] G. Savard, *Contribution a la programmation mathematique a deux niveaux*, Ph.D. thesis, Ecole Polytechnique, Univ. de Montreal, Québec, Canada, 1989.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 783–799, July 1993

THE ACCURACY OF FLOATING POINT SUMMATION*

NICHOLAS J. HIGHAM†

Abstract. The usual recursive summation technique is just one of several ways of computing the sum of n floating point numbers. Five summation methods and their variations are analyzed here. The accuracy of the methods is compared using rounding error analysis and numerical experiments. Four of the methods are shown to be special cases of a general class of methods, and an error analysis is given for this class. No one method is uniformly more accurate than the others, but some guidelines are given on the choice of method in particular cases.

Key words. floating point summation, rounding error analysis, orderings

AMS subject classifications. primary 65G05, secondary 65B10

1. Introduction. Sums of floating point numbers are ubiquitous in scientific computing. They occur when evaluating inner products, means, variances, norms, and all kinds of nonlinear functions. Although, at first sight, summation might appear to offer little scope for algorithmic ingenuity, the usual "recursive summation" (with various orderings) is just one of several possible techniques. Most of the other techniques have been derived with the aim of achieving greater accuracy of the computed sum, but pairwise summation has the advantage of being particularly well suited to parallel computation.

In this paper we examine a variety of methods for floating point summation, with the aim of answering the question, "which methods achieve the best accuracy?" Several authors have used error analysis to compare summation methods; see, for example, [1], [2], [32], and [38]. Here we give a more comprehensive treatment that highlights the relationships between different methods; in particular, we give an error analysis for a general class of methods that includes most of the specific summation methods as special cases.

This work was motivated by two applications in which the choice of summation method has been found to have an important influence on the performance of a numerical method.

(1) In [24], Lasdon et al. derive an algorithm for solving an optimization problem that arises in the design of sonar arrays. The authors state [24, p. 145] that "the objective gradient ∇f in (4.1) is a sum of M terms. In problems with $M = 284$ and $n = 42$, the GRG2 optimizer encountered difficulties which stem from inaccuracies in ∇f.... We hypothesized that this was due to roundoff error resulting from cancellation of terms in ∇f of approximately equal magnitudes and opposite signs. These problems were eliminated by accumulating separately positive and negative terms (for each component of ∇f) in the sum (4.1), adding them together only after all M terms had been processed."

(2) Dixon and Mills [7] applied a quasi-Newton method to the extended Rosenbrock function

$$F(x_1, x_2, \dots, x_n) = \sum_{i=1}^{n/2} \bigl(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2\bigr). \tag{1.1}$$

This function and its derivatives possess certain symmetries; for example, $F(a, b, c, d) = F(c, d, a, b)$ when $n = 4$. It is observed in [7] that expected symmetries in the search

*Received by the editors August 12, 1991; accepted for publication (in revised form) July 10, 1992.

†Department of Mathematics, University of Manchester, Manchester, M13 9PL, United Kingdom (na.nhigham@na-net.ornl.gov).

direction and Hessian approximation are lost in practice, resulting in more iterations for convergence of the quasi-Newton method than are predicted theoretically. Dixon and Mills attribute the loss of symmetry to rounding errors in the evaluation of certain inner products, which can cause identities such as the one quoted above to fail in floating point arithmetic. They restore symmetry (and thus reduce the number of iterations) by using a special summation algorithm when evaluating inner products: their algorithm evaluates $\sum_{i=1}^{n} x_i$ by sorting the x_i, dividing them into a list of negative numbers and a list of nonnegative numbers, and then repeatedly forming the sum of the largest nonnegative and most negative elements and placing the sum into the appropriate list, in order.

We return to these two applications in §7.

The five main summation methods that we consider are defined and analyzed in §§2 and 3. For the error analysis we will make use of the standard model of floating point arithmetic, in which u is the unit roundoff:

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \qquad |\delta| \le u, \quad \text{op } = +,-,*,/. \tag{1.2}$$

This model is violated by machines that lack a guard digit, so we explain in §5 how our analysis has to be modified to accommodate such machines. We will assume that no floating point underflows occur; how to modify error analyses to allow for underflow is described by Demmel in [6]. An excellent tutorial on many aspects of floating point arithmetic is given by Goldberg [9].

In §4 we summarize some existing results on statistical estimates of accuracy of summation methods. Numerical experiments are presented in §6 and conclusions are given in §7.

2. Orderings of recursive summation.

Our task is to evaluate $S_n = \sum_{i=1}^{n} x_i$, where $x_1, \dots, x_n$ are real numbers. In this section we consider the standard recursive summation technique in which S_n is evaluated according to

```
$s = 0$
for $i = 1{:}\,n$
    $s = s + x_i$
end
```

In general, each different ordering of the x_i will yield a different computed sum $\widehat{S}_n$ in floating point arithmetic, and it is of interest to determine how the ordering affects the error

$$E_n = \widehat{S}_n - S_n.$$

To begin, we make no assumption on the ordering and obtain a standard bound for E_n. By a direct application of the model (1.2) we have, with $S_k = \sum_{i=1}^{k} x_i$,

$$\widehat{S}_k = fl(\widehat{S}_{k-1} + x_k) = (\widehat{S}_{k-1} + x_k)(1+\delta_k), \qquad |\delta_k| \le u, \quad k = 2{:}\,n. \tag{2.1}$$

By repeated use of this relation it follows that

$$\widehat{S}_n = (x_1 + x_2)\prod_{k=2}^{n}(1+\delta_k) + \sum_{i=3}^{n} x_i \prod_{k=i}^{n}(1+\delta_k). \tag{2.2}$$

To simplify the product terms we use the result that if $|\delta_i| \le u$ for $i = 1{:}\,n$ then

$$\prod_{i=1}^{n}(1+\delta_i) = 1 + \theta_n, \quad \text{where} \quad |\theta_n| \le \frac{nu}{1-nu} \equiv \gamma_n.$$

Thus we rewrite (2.2) as

$$\widehat{S}_n = (x_1 + x_2)(1 + \theta_{n-1}) + \sum_{i=3}^{n} x_i(1 + \theta_{n-i+1}), \tag{2.3}$$

which yields

$$|E_n| = |(x_1 + x_2)\theta_{n-1} + \sum_{i=3}^{n} x_i\theta_{n-i+1}| \tag{2.4}$$

$$\leq (|x_1| + |x_2|)\gamma_{n-1} + \sum_{i=3}^{n} |x_i|\gamma_{n-i+1}. \tag{2.5}$$

Note that x_1 and x_2 have the same factor γ_{n-1}, because they play identical roles in the summation. The bound (2.5) is essentially the same as the one derived by Wilkinson in [42, p. 323] and [43, p. 17]. As Wilkinson notes in [43, p. 17], the upper bound in (2.5) depends on the order of summation, and the bound is minimized if the x_i are arranged in order of increasing absolute value. We emphasize that this ordering minimizes an error *bound* and not necessarily the actual error (this is illustrated later in this section, and by numerical examples in §6). We can weaken (2.5) to obtain the bound

$$|E_n| \leq \gamma_{n-1} \sum_{i=1}^{n} |x_i| = (n-1)u \sum_{i=1}^{n} |x_i| + O(u^2), \tag{2.6}$$

which is independent of the ordering. It is natural to regard satisfaction of (2.6) as a minimal requirement of any summation method; in fact, all the methods we will examine do satisfy this bound.

We can rewrite (2.6) as the relative error bound

$$\frac{|\widehat{S}_n - S_n|}{|S_n|} \leq \gamma_{n-1} \frac{\sum_{i=1}^{n} |x_i|}{|S_n|} \equiv \gamma_{n-1} R_n.$$

In the special case where $x_i \geq 0$ for all u, $R_n \equiv 1$, and the relative error has a bound of order nu, but if $\sum_{i=1}^{n} |x_i| \gg |\sum_{i=1}^{n} x_i|$ we cannot guarantee that the relative error is small. The quantity R_n is easily seen to be the condition number of summation when perturbations $x_i \to x_i + \Delta x_i$ are measured by $\max_i |\Delta x_i|/|x_i|$.

Recursive summation by order of increasing absolute value can be improved upon in two possible ways. First, a method may satisfy a bound of the form (2.6) but with a constant smaller than γ_{n-1}. Second, a method may satisfy a bound that in the worst case is no better than (2.6), but the method might be expected to yield a more accurate computed sum for particular classes of $\{x_i\}$. In the rest of this section we consider two alternative orderings, which fall into the second category.

First, we derive a sharper error bound. From (2.1) we see that the error introduced on the kth step of the summation is $(\widehat{S}_{k-1} + x_k)\delta_k = \widehat{S}_k\delta_k/(1 + \delta_k)$. Summing these individual errors we find

$$E_n = \sum_{k=2}^{n} \widehat{S}_k \frac{\delta_k}{1 + \delta_k}, \tag{2.7}$$

which shows that, to first order, the overall error is the sum of the $n-1$ relative rounding errors weighted by the partial sums. We obtain the bound

$$|E_n| = |\widehat{S}_n - S_n| \leq \frac{u}{1-u} \sum_{k=2}^{n} |\widehat{S}_k|. \tag{2.8}$$

This bound involves the computed partial sums (excluding $\widehat{S}_1 = x_1$) but not the individual terms x_i. If we weaken (2.8) by bounding $|\widehat{S}_k|$ in terms of $|x_1|, |x_2|, \ldots, |x_k|$, then we recover (2.5), to within $O(u^2)$.

The bound (2.8) suggests the strategy of ordering the x_i so as to minimize $\sum_{k=2}^{n} |\widehat{S}_k|$. This is a combinatorial optimization problem that is too expensive to solve in the context of summation. A reasonable compromise is to determine the ordering sequentially by minimizing, in turn, $|x_1|, |\widehat{S}_2|, \ldots, |\widehat{S}_{n-1}|$. This ordering strategy, which we denote by Psum, can be implemented with $O(n \log n)$ comparisons. The principal difference between the Psum and increasing orderings is that the Psum ordering is influenced by the signs of the x_i, while the increasing ordering is independent of the signs. If all the x_i have the same sign then the two orderings are identical.

It is easy to show by example that the bounds (2.8), (2.5) and (2.6) are nearly attainable. Following Wilkinson [43, p. 19] we assume $u = 2^{-t}$, set $n = 2^r$ ($r \ll t$), and define

$$\begin{aligned} x(1) &= 1, \\ x(2) &= 1 - 2^{-t}, \\ x(3{:}4) &= 1 - 2^{1-t}, \\ x(5{:}8) &= 1 - 2^{2-t}, \\ &\vdots \\ x(2^{r-1}+1{:}2^r) &= 1 - 2^{r-1-t}. \end{aligned}$$

Then in the $(i-1)$st floating point addition the "2^{k-t}" portion of x_i does not propagate into the sum;[1] thus there is an error of 2^{k-t} and $\widehat{S}_i = i$. The total error is

$$2^{-t}(1 + 2^2 + 2^4 + \cdots + 2^{2(r-1)}) = 2^{-t}\frac{4^r - 1}{3},$$

while the upper bound of (2.6) is

$$\frac{(n-1)u}{1-(n-1)u} \sum_{i=1}^{n} |x_i| \le \frac{2^r 2^{-t}}{1 - 2^r 2^{-t}} 2^r \approx 2^{-t} 4^r,$$

which agrees with the actual error to within a factor 3; thus the smaller upper bounds of (2.5) and (2.8) are also correct to within this factor. The example just quoted is, of course, a very special one, and as Wilkinson [43, p. 20] explains, "in order to approach the upper bound as closely as this, not only must each error take its maximum value, but all the terms must be almost equal."

Next, we consider ordering the x_i by decreasing absolute value. For the summation of positive numbers this ordering has little to recommend it. The bound (2.8) is no smaller, and potentially much larger, than for the increasing ordering (the same is true for the weaker bound (2.5)). Furthermore, in a sum of positive terms that vary widely in magnitude, the decreasing ordering may not allow the smaller terms to contribute to the sum (which is why $\sum_{i=1}^{n} 1/i$ "converges" in floating point arithmetic as $n \to \infty$). However, consider the example with $n = 4$ and

$$x = [1, \quad M, \quad 2M, \quad -3M], \tag{2.9}$$

[1]We assume in this example that the floating point arithmetic uses round to nearest with ties broken by rounding to an even last bit or rounding away from zero.

where M is a power of the machine base and is so large that $fl(1+M) = M$ (thus $M \geq u^{-1}$). The three orderings considered so far produce the following results:

$$\begin{aligned}
\text{Increasing:} \quad & \widehat{S}_n = fl(1+M+2M-3M) = 0, \\
\text{Psum:} \quad & \widehat{S}_n = fl(1+M-3M+2M) = 0, \\
\text{Decreasing:} \quad & \widehat{S}_n = fl(-3M+2M+M+1) = 1.
\end{aligned}$$

Thus the decreasing ordering sustains no rounding errors and produces the exact answer, while both the increasing and Psum orderings yield computed sums with relative error 1. The reason why the decreasing ordering performs so well in this example is that it adds the "1" after the inevitable heavy cancellation has taken place, rather than before, and so retains the important information in this term.

If we evaluate the term $\mu = \sum_{k=2}^{n} |\widehat{S}_k|$ in the error bound (2.8) for the example (2.9) we find

$$\text{Increasing: } \mu = 4M, \qquad \text{Psum: } \mu = 3M, \qquad \text{Decreasing: } \mu = M+1,$$

so (2.8) "predicts" that the decreasing ordering will produce the most accurate answer, but the bound it provides is extremely pessimistic since there are no rounding errors in this instance. This example illustrates the main weakness of bounds from a rounding error analysis: they represent the worst case and so do not account for the possibility that rounding errors may cancel or be smaller than expected.

Extrapolating from this example, we conclude that the decreasing ordering is likely to yield greater accuracy than the increasing or Psum orderings whenever there is heavy cancellation in the sum, that is, whenever $|\sum_{i=1}^{n} x_i| \ll \sum_{i=1}^{n} |x_i|$. A numerical example that illustrates this assertion is given in §6 (see Table 6.1).

3. Other methods. In this section we consider in detail four more summation methods. The first three of these methods, together with recursive summation, have the following general form: with $T_k \equiv x_k$, $k = 1{:}\,n$, they perform $n-1$ additions

$$T_k = T_{k_1} + T_{k_2}, \quad k_1 < k_2 < k, \quad k = n+1{:}\,2n-1, \tag{3.1}$$

yielding $S_n = T_{2n-1}$. In recursive summation, $k_1 \leq n$ in each instance of (3.1), but for the other methods at least one addition involves two previously computed sums. A useful expression for the error in this general class of summation methods can be derived as follows. The computed quantities $\widehat{T}_k$ satisfy

$$\widehat{T}_k = (\widehat{T}_{k_1} + \widehat{T}_{k_2})(1+\delta_k), \quad |\delta_k| \leq u, \quad k = n+1{:}\,2n-1. \tag{3.2}$$

The local error introduced in forming $\widehat{T}_k$ is $(\widehat{T}_{k_1} + \widehat{T}_{k_2})\delta_k = \widehat{T}_k\delta_k/(1+\delta_k)$, so overall we have

$$\widehat{S}_n - S_n = \sum_{k=n+1}^{2n-1} \widehat{T}_k \frac{\delta_k}{1+\delta_k}. \tag{3.3}$$

The smallest possible error bound is therefore

$$|E_n| \leq \frac{u}{1-u} \sum_{k=n+1}^{2n-1} |\widehat{T}_k|. \tag{3.4}$$

It is easy to see that $|\widehat{T}_k| \le \sum_{i=1}^{n} |x_i| + O(u)$ for each k, and so we also have the weaker bound

$$|E_n| \le (n-1)u \sum_{i=1}^{n} |x_i| + O(u^2). \tag{3.5}$$

Note that in the case of recursive summation (3.3)–(3.5) are the same as (2.6)–(2.8). Finally, we note in passing that from (3.2) there follows a backward error result which shows that $\widehat{S}_n$ is the exact sum of terms $x_i(1+\theta_i)$, where $|\theta_i| = O(u)$.

The first method we consider is pairwise summation (also known as cascade summation), which was first discussed by McCracken and Dorn [29, pp. 61–63], Babuška [1], and Linz [27]. In this method the x_i are summed in pairs,

$$y_i = x_{2i-1} + x_{2i}, \quad i = 1 : \left[\tfrac{n}{2}\right] \quad (y_{[(n+1)/2]} = x_n \text{ if } n \text{ is odd}),$$

and this pairwise summation process is repeated recursively on the y_i, $i = 1 : [(n+1)/2]$. The sum is obtained in $\lceil \log_2 n \rceil$ stages. For $n = 6$, for example, pairwise summation forms

$$S_6 = \big((x_1 + x_2) + (x_3 + x_4)\big) + (x_5 + x_6).$$

Pairwise summation is attractive in parallel settings, because each of the $\lceil \log_2 n \rceil$ stages can be done in parallel [13, §5.2.2]. Caprani [4] shows how to implement the method on a serial machine using temporary storage of size $\lceil \log_2 n \rceil$ (without overwriting the x_i).

The error expression (3.3) holds for pairwise summation, but it is easy to derive a useful error bound independently. Assume for simplicity that $n = 2^r$. Unlike in recursive summation each addend takes part in the same number of additions, $\log_2 n$. Therefore, analogously to (2.2), we have a relation of the form

$$\widehat{S}_n = \sum_{i=1}^{n} x_i \prod_{k=1}^{\log_2 n} (1 + \delta_k^{(i)}), \qquad |\delta_k^{(i)}| \le u,$$

which leads to the bound

$$|E_n| \le \gamma_{\log_2 n} \sum_{i=1}^{n} |x_i|. \tag{3.6}$$

This is a smaller bound than (2.6) for recursive summation, since it is proportional to $\log_2 n$ rather than n. However, in special cases the bound (2.5) for recursive summation can be smaller than (3.6). For example, if $x_i = 1/i^3$, the bound (3.6) is

$$|E_n| \le 1.20 \log_2 n\, u + O(u^2) \tag{3.7}$$

(using $\sum_{i=1}^{n} 1/i^3 \approx \sum_{i=1}^{\infty} 1/i^3 \approx 1.20$), while for the increasing ordering (2.5) becomes

$$|E_n| \le u \sum_{i=1}^{n} \frac{1}{(n-i+1)^3}(n-i+1) + O(u^2) = u \sum_{i=1}^{n} \frac{1}{i^2} + O(u^2) \approx 1.64u + O(u^2)$$

(using $\sum_{i=1}^{n} 1/i^2 \approx \sum_{i=1}^{\infty} 1/i^2 = \pi^2/6 \approx 1.64$), and so pairwise summation has the larger error bound, by a factor $\approx \log_2 n$. (Expression (3.3) does not enable us to improve on the factor $\log_2 n$ in (3.7).)

In [35] an "insertion adder" is proposed for the summation of positive numbers. This method can be applied equally well to arbitrary sums. First, the x_i are sorted by order of increasing magnitude. Then $x_1 + x_2$ is formed, and the sum is inserted into the list $x_2, \dots, x_n$, maintaining the increasing order. The process is repeated recursively until the final sum is obtained. The motivation given in [35] for this strategy is that it tends to encourage the additions to be between numbers of similar magnitude. It can be argued that such additions are to be preferred, because they retain more of the information in the addends (by comparison, "large" plus "small" may lose many significant digits from "small"). A more convincing explanation of the insertion strategy is that it attempts to minimize, one at a time, the absolute values of the terms $\widehat{T}_{n+1}, \dots, \widehat{T}_{2n-1}$ in the error expression (3.3). Indeed, if the x_i are all positive the insertion method minimizes the bound (3.4) over all methods of the form (3.1).

In particular cases the insertion method reduces to earlier methods. For example, if $x_i = 2^i$, the insertion method is equivalent to recursive summation, since the insertion is always to the bottom of the list:

$$1\,2\,4\,8 \quad \to \quad \underline{3}\,4\,8 \quad \to \quad \underline{7}\,8 \quad \to \quad 15.$$

On the other hand, if $1 \le x_1 < x_2 < \cdots < x_n \le 2$, every insertion is to the end of the list, and the method is equivalent to pairwise summation if n is a power of 2; for example, if $0 < \epsilon < \frac{1}{2}$,

$$1,\ 1+\epsilon,\ 1+2\epsilon,\ 1+3\epsilon \quad \to \quad 1+2\epsilon,\ 1+3\epsilon,\ \underline{2+\epsilon} \quad \to \quad 2+\epsilon,\ \underline{2+5\epsilon} \quad \to \quad 4+6\epsilon.$$

The next method we consider is the one used in [24], as quoted in the Introduction. This method can be derived by the following specious reasoning: "A major source of inaccuracy in floating point summation is cancellation when numbers of nearly equal magnitude and opposite sign are added. To minimize the amount of cancellation we can accumulate the sum of the positive numbers, S_+, and the sum of the negative numbers, S_-, separately, and then form $S_n = S_+ + S_-$." There are two flaws in this argument. First, this "$+/-$" method does not reduce the amount of cancellation—it simply concentrates all the cancellation into one step. Second, cancellation is not a bad thing per se; the problem with cancellation is that it brings into prominence any loss of significant digits suffered earlier in the calculation (and it also brings into prominence any uncertainty in the data). Indeed, nearly equal floating point numbers are always subtracted *exactly* (assuming the presence of a guard digit)—it is any (relative) uncertainty in those numbers that is magnified. For an excellent and more detailed discussion of cancellation, we refer the reader to [34, pp. 25–29].

The $+/-$ method is of the form (3.1) (assuming that S_+ and S_- are computed using one of the methods discussed so far) and it is easy to see that it maximizes $\max_k |T_k|$ over all methods of this form. Moreover, when $\sum_{i=1}^n |x_i| \gg |\sum_{i=1}^n x_i|$ the value of $\max_k |T_k|$ tends to be much larger for the $+/-$ method than for the other methods we have considered. For example, if $n = 2m$ and the x_i are the values $\{-1, 1, -2, 2, \dots, -m, m\}$ then

$$|S_+| = |S_-| = |T_{2n-2}| = \sum_{i=1}^{m} k = m(m+1)/2,$$

whereas for recursive summation with the increasing ordering, $|T_k| \le m$ for all k. Despite this weakness, if S_+ and S_- are computed by recursive summation with the increasing ordering then the $+/-$ method satisfies a bound very similar to (2.5): if

$$x_p \le x_{p-1} \le \cdots \le x_1 < 0 \le x_{p+1} \le \cdots \le x_n,$$

then it is straightforward to derive the bound

$$|E_n| \le \sum_{i=1}^{p} \gamma_{p-i+1}|x_i| + \sum_{i=p+1}^{n} \gamma_{n-i+1}|x_i| + \frac{u}{1-u}|\widehat{S}_n|. \tag{3.8}$$

In summary, the $+/-$ method appears to have no advantages over the other methods considered here, and in cases where there is heavy cancellation in the sum it can be expected to be the least accurate method.

The final method that we examine has an interesting background. In 1951, Gill [8] noticed that the rounding error in the sum of two numbers could be estimated by subtracting one of the numbers from the sum, and he made use of this estimate in a Runge–Kutta code in a program library for the EDSAC computer. Gill's estimate is valid for fixed point arithmetic only. Kahan [16] and Møller [31] both extended the idea to floating point arithmetic. Møller shows how to estimate $a + b - fl(a + b)$ in chopped arithmetic, while Kahan uses a slightly simpler estimate to derive a "compensated summation" method for computing $\sum_{i=1}^{n} x_i$. The use of Kahan's method with a Runge–Kutta formula is described in [41] (see also the experiments in [26]).

The estimate used by Kahan is perhaps best explained with the aid of a diagram. Let a and b be floating point numbers with $|a| \ge |b|$, let $\widehat{s} = fl(a + b)$, and consider Fig. 3.1, which uses boxes to represent the mantissas of a and b. The figure suggests that if we evaluate

$$e = -\big[\big((a+b) - a\big) - b\big] = \big(a - \widehat{s}\big) + b$$

in floating point arithmetic, in the order indicated by the parentheses, then the computed $\widehat{e}$ will be a good estimate of the error $(a + b) - \widehat{s}$. In fact, for rounded floating point arithmetic in base 2, we have

$$a + b = \widehat{s} + \widehat{e}, \tag{3.9}$$

that is, the computed $\widehat{e}$ represents the error exactly. This result (which does not hold for all bases) is proved by Dekker [5, Thm. 4.7], Knuth [22, Thm. C, p. 221], and Linnainmaa [26, Thm. 3]. Note that there is no point in computing $fl(\widehat{s} + \widehat{e})$, since $\widehat{s}$ is already the best floating point representation of $a + b$!

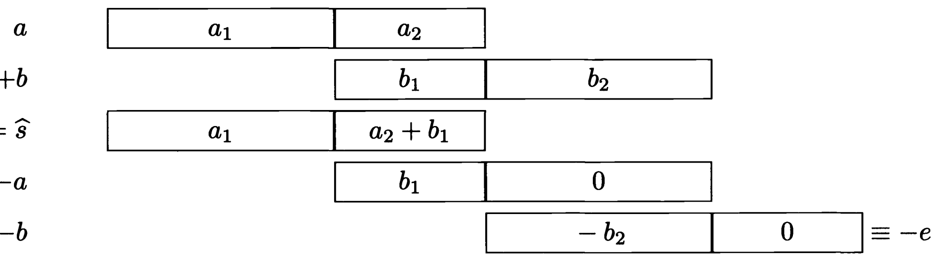


FIG. 3.1. *Recovering the rounding error.*

Kahan's compensated summation method employs the correction e on every step of a recursive summation. After each partial sum is formed, the correction is computed and immediately added to the next term x_i before that term is added to the partial sum.

Thus the idea is to capture the rounding errors and feed them back into the summation. The method may be written as follows.

```
$s = 0$; $e = 0$
for $i = 1{:}\,n$
    $temp = s$
    $y = x_i + e$
    $s = temp + y$
    $e = (temp - s) + y$
end
```

The compensated summation method has two weaknesses: $\widehat{e}$ is not necessarily the exact correction, since (3.9) is based on the assumption that $|a| \ge |b|$, and the addition $y = x_i + e$ is not performed exactly. Nevertheless, the use of the corrections brings a benefit in the form of an improved error bound. Knuth [22, Ex. 19, pp. 229, 572–573] shows that the computed sum $\widehat{S}_n$ satisfies

$$\widehat{S}_n = \sum_{i=1}^{n}(1+\mu_i)x_i, \qquad |\mu_i| \le 2u + O(nu^2), \tag{3.10}$$

which is an almost ideal backward error result (a more detailed version of Knuth's proof is given in [9]).

In [17] and [18], Kahan describes a variation of compensated summation in which the final sum is also corrected (thus "$s = s+e$" is appended to the algorithm above). Kahan states in [17] and proves in [18] that (3.10) holds with the stronger bound $|\mu_i| \le 2u+ O\big((n-i+1)u^2\big)$; note that with this bound for $|\mu_i|$, (3.10) is essentially (2.3) with the n dependency transferred from the u term to the u^2 term. The proofs of (3.10) given by Knuth and Kahan are similar, and involve a subtle induction using the model (1.2).

The forward error bound corresponding to (3.10) is

$$|E_n| \le \big(2u + O(nu^2)\big)\sum_{i=1}^{n}|x_i|. \tag{3.11}$$

As long as $nu \le 1$, the constant in this bound is independent of n, and so the bound is a significant improvement over the bounds (2.6) for recursive summation and (3.6) for pairwise summation. Note, however, that if $\sum_{i=1}^{n}|x_i| \gg |\sum_{i=1}^{n} x_i|$, compensated summation is not guaranteed to yield a small relative error.

Another version of compensated summation is described in [14], [15], [21], [32], and [33]. Here, instead of immediately feeding each correction back into the summation, the corrections are accumulated by recursive summation and then the global correction is added to the computed sum. For this version of compensated summation it is shown in [21] and [32] that

$$\widehat{S}_n = \sum_{i=1}^{n}(1+\mu_i)x_i, \qquad |\mu_i| \le 2u + n^2u^2, \tag{3.12}$$

provided $nu \le 0.1$; this is weaker than (3.10) in that the second-order term has an extra factor n. If $n^2u \le 0.1$ then in (3.12), $|\mu_i| \le 2.1u$. In [14], it is shown that by using a divide and conquer implementation of compensated summation the range of n for which $|\mu_i| \le cu$ holds in (3.12) can be extended, at the cost of a slight increase in the size of the constant c.

Finally, we mention briefly two further classes of algorithms. The first builds the sum in a series of accumulators, which are themselves added to give the sum. As originally described in [44], each accumulator holds a partial sum lying in a different interval. Each term x_i is added to the lowest level accumulator; if that accumulator overflows it is added to the next higher one and then reset to zero, and this cascade continues until no overflow occurs. Modifications of Wolfe's algorithm are presented in [28] and [36]. Malcolm [28] gives a detailed error analysis to show that his method achieves a relative error of order u. A drawback of the algorithm is that it is strongly machine dependent. An interesting and crucial feature of Malcolm's algorithm is that on the final step the accumulators are summed by recursive summation in order of *decreasing* absolute value, which in this particular situation precludes severe loss of significant digits and guarantees a small relative error.

Another class of algorithms, referred to as "distillation algorithms" by Kahan [19], iteratively constructs floating point numbers $x_1^{(k)}, \ldots, x_n^{(k)}$ such that $\sum_{i=1}^n x_i^{(k)} = \sum_{i=1}^n x_i$, terminating when $x_n^{(k)}$ approximates $\sum_{i=1}^n x_i$ with relative error at most u. Kahan states that these algorithms appear to have average run times of order at least $n \log n$. See [3], [19], [25] and [23] for further details and references.

4. Statistical estimates of accuracy. As we have noted, rounding error bounds can be very pessimistic, because they account for the worst-case propagation of errors. An alternative way to compare summation methods is through statistical estimates of the error, which may be more representative of the average case. A statistical analysis of three summation methods has been given by Robertazzi and Schwartz [35] for the case of nonnegative x_i. They assume that the relative errors in floating point addition are statistically independent and have zero mean and finite variance σ^2. Two distributions of nonnegative x_i are considered: the uniform distribution on $[0, 2\mu]$, and the exponential distribution with mean μ. Making various simplifying assumptions Robertazzi and Schwartz estimate the mean square error (that is, the variance of the absolute error) of the computed sums from recursive summation with random, increasing, and decreasing orderings, and from insertion summation and pairwise summation. Their results for the summation of n numbers are given in Table 4.1.

TABLE 4.1
Mean square errors.

Distribution	Increasing	Random	Decreasing	Insertion	Pairwise
Unif$(0, 2\mu)$	$0.20\mu^2n^3\sigma^2$	$0.33\mu^2n^3\sigma^2$	$0.53\mu^2n^3\sigma^2$	$2.6\mu^2n^2\sigma^2$	$2.7\mu^2n^2\sigma^2$
Exp(μ)	$0.13\mu^2n^3\sigma^2$	$0.33\mu^2n^3\sigma^2$	$0.63\mu^2n^3\sigma^2$	$2.6\mu^2n^2\sigma^2$	$4.0\mu^2n^2\sigma^2$

The results show that for recursive summation the ordering affects only the constant in the mean square error, with the increasing ordering having the smallest constant and the decreasing ordering the largest; since the x_i are nonnegative, this is precisely the ranking given by the rounding error bound (2.8). The insertion and pairwise summation methods have mean square errors proportional to n^2 rather than n^3 for recursive summation, and the insertion method has a smaller constant than pairwise summation. This is also consistent with the rounding error analysis, in which for nonnegative x_i the insertion method satisfies an error bound no larger than pairwise summation, and the latter method has an error bound with a smaller constant than for recursive summation ($\log_2 n$ versus n).

5. No guard digit model. The model (1.2) on which our error analysis is based is not valid on machines that lack a guard digit in addition, notable examples of which are Cray computers. On Cray computers, subtracting any power of 2 from the next smaller floating point number gives an answer that is either a factor of 2 too large or is zero, so the expression $fl(x+y) = (x+y)(1+\delta)$ holds with $|\delta| = 1$ but not with $|\delta| = O(u)$ [20]. For machines without a guard digit we have to use the weaker model [43, p. 12]

$$fl(x \pm y) = x(1+\alpha) \pm y(1+\beta), \qquad |\alpha|, |\beta| \le u. \tag{5.1}$$

We now summarize the effect on the rounding error analysis of using (5.1) in place of (1.2). The equality (2.4) remains valid provided we replace $(x_1+x_2)\theta_{n-1}$ by $x_1\theta_{n-1} + x_2\theta'_{n-1}$, so (2.5) and (2.6) are unchanged. The error expression (2.7) has to be replaced by

$$E_n = \sum_{k=2}^{n} (\widehat{S}_{k-1}\alpha_k + x_i\beta_k), \qquad |\alpha_k|, |\beta_k| \le u, \tag{5.2}$$

and so the analog of (2.8) is

$$|E_n| \le u \sum_{k=2}^{n} (|\widehat{S}_{k-1}| + |x_k|), \tag{5.3}$$

which is bounded above by $3u\sum_{k=1}^{n}|\widehat{S}_k| + O(u^2)$. Notice that (5.3) contains the term $\widehat{S}_1 = x_1$, which is not present in (2.8). The error expression (3.3) has to be replaced by an expression analogous to (5.2), and in (3.5) the factor $n-1$ has to be replaced by n. The bound (3.6) for pairwise summation remains valid under the no guard digit model, while in the bound (3.8) for the $+/-$ method we have to replace $u/(1-u)|\widehat{S}_n|$ by $u(|\widehat{S}_+| + |\widehat{S}_-|)$, which is bounded by $u\sum_{i=1}^{n}|x_i| + O(u^2)$.

Neither the correction formula (3.9) nor the result (3.10) for compensated summation holds under the no guard digit model. Indeed, Kahan [20] constructs an example where compensated summation fails to achieve (3.11) on Cray machines, but he states that such failure is extremely rare. In [17] and [18], Kahan gives a modification of the compensated summation algorithm in which the assignment "$e = (temp - s) + y$" is replaced by

```
$f = 0$
if sign($temp$) = sign($y$), $f = (0.46 * s - s) + s$, end
$e = ((temp - f) - (s - f)) + y$
```

Kahan shows in [18] that the modified algorithm achieves (3.10) "on all North American machines with floating hardware" and explains that, "the mysterious constant 0.46, which could perhaps be any number between 0.25 and 0.50, and the fact that the proof requires a consideration of known machines designs, indicate that this algorithm is not an advance in computer science."

6. Numerical experiments. In this section we describe some numerical experiments that give further insight into the accuracy of summation methods. All the experiments were done using MATLAB [30], which uses IEEE standard double precision arithmetic with unit roundoff $u \approx 1.1 \times 10^{-16}$.

First, we illustrate the behavior of the methods on four classes of data $\{x_i\}$ chosen a priori. In these tests we simulated single precision arithmetic of unit roundoff

$u_{SP} = 2^{-23} \approx 1.2 \times 10^{-7}$ by rounding the result of every arithmetic operation to 23 significant bits. We formed an approximation to the exact answer S_n by summing the single precision numbers x_i in double precision by recursive summation; in each case $nu \sum_{i=1}^{n} |x_i| < u_{SP}|S_n|$, so (2.6) guarantees that this approximation is correct to single precision. We give results for recursive summation with the original (Orig.), increasing (Inc.), decreasing (Dec.), and Psum orderings, and for the insertion (Ins.) method, the $+/-$ method (with S_+ and S_- computed by recursive summation with the increasing ordering), pairwise summation with the increasing ordering (Pair.), and compensated summation (Comp.).

The numbers reported are the relative error $|\widehat{S}_n - S_n|/|S_n|$, together with information that indicates the sharpness of the bounds. In square brackets is the value $T = \sum_{k=n+1}^{2n-1} |\widehat{T}_k|$ in (3.4), for all methods except compensated summation. In parentheses is the ratio $R = |\widehat{S}_n - S_n|/(u_{SP}\sum_{i=1}^{n} |x_i|)$, which, according to the error analyses, is certainly bounded (to first order) by n for recursive summation and the insertion and $+/-$ methods, $\log_2 n$ for pairwise summation, and 2 for compensated summation. The quantities T and R reveal how close the strongest and weakest of the error bounds are to being equalities.

(1) In the first example, x_i is the ith term in the Taylor series expansion of e^{-x} about the origin, with $x = 2\pi$ (this series provides the classic example of "catastrophic cancellation" [37]). Results for $n = 64$ are given in Table 6.1. In this example, recursive summation with the decreasing ordering yields by far the best accuracy. There is severe cancellation in the sum and the decreasing ordering allows the terms of smallest modulus to contribute fully to the computed sum; in the other methods the small terms are "swamped" by the large terms. The error bounds do not reflect the merit of the decreasing ordering, because the T terms (in square brackets in the table) are of similar magnitude for the first four methods. Note also that compensated summation produces no improvement over recursive summation with the original ordering, and the $+/-$ method yields one less correct significant figure than all the other methods (as predicted by the T values).

TABLE 6.1
x_i from $e^{-2\pi}$ expansion. $|\sum_{i=1}^{n} x_i| / \sum_{i=1}^{n} |x_i| = 3.48e-6$.

n	Orig.	Inc.	Dec.	Psum	Pair.	Ins.	$+/-$	Comp.
64	5.11e−4	2.27e−3	1.85e−7	2.27e−3	1.41e−4	2.27e−3	1.86e−2	5.11e−4
	[2.68e2	2.97e2	2.97e2	2.85e2	8.68e1	2.97e2	1.34e3]	
	(1.49e−2	6.64e−2	5.40e−6	6.64e−2	4.13e−3	6.64e−2	5.44e−1	1.49e−2)

(2) In this example the x_i are random numbers from the Normal(0,1) distribution and we report the results for $n = 2048$ and 4096 in Table 6.2. There is cancellation in both sums, although not as much as in the first example. Here the Psum ordering is clearly the best and the $+/-$ ordering the worst, and this is reflected in the T values.

The next two tests involve positive x_i, for which all the methods are guaranteed to produce a relative error no larger than $f(n)u$, where $f(n) \le n$ depends on the method. (Note that for positive x_i, "Psum $\equiv +/- \equiv$ Inc.")

(3) We take $x_i = 1/i^2$ and examine how the errors vary with n for recursive summation with the decreasing and increasing orderings. Results for $n = 500, 1000, \ldots, 5000$ are given in Table 6.3. As would be expected in view of the error bounds of §2, the decreasing ordering provides much lower accuracy than the increasing ordering when n is large.

TABLE 6.2
x_i from Normal (0,1) *distribution.* $|\sum_{i=1}^n x_i|/\sum_{i=1}^n |x_i| = 8.08e-3 (n = 2048), 3.48e-3 (n = 4096)$.

n	Orig.	Inc.	Dec.	Psum	Pair.	Ins.	+/−	Comp.
2048	7.47e−6	3.32e−6	7.17e−6	6.82e−8	6.60e−7	5.12e−7	1.20e−4	2.28e−7
	[3.06e4	1.53e4	2.65e4	8.26e2	2.87e3	2.32e3	4.80e5]	
	(5.06e−1	2.25e−1	4.86e−1	4.62e−3	4.47e−2	3.47e−2	8.15e0	1.54e−2)
4096	8.06e−6	1.04e−5	1.84e−6	2.66e−8	1.38e−7	6.87e−7	3.68e−4	1.92e−7
	[6.69e4	4.74e4	4.28e4	1.68e3	5.70e3	5.38e3	2.02e6]	
	(2.35e−1	3.04e−1	5.38e−2	7.76e−4	4.04e−3	2.00e−2	1.07e1	5.59e−3)

TABLE 6.3
$x_i = 1/i^2$.

n	500	1000	2000	3000	4000	5000
Inc.	1.04e−7	1.01e−7	1.74e−8	5.22e−8	1.36e−7	3.90e−8
Dec.	3.31e−7	6.24e−7	5.64e−6	2.30e−5	2.77e−5	5.81e−5

(4) In this example the numbers x_i are equally spaced on $[1, 2]$. We tried various $n \leq 4096$ and did not observe a great difference between the increasing and decreasing orderings; this is to be expected since the x_i vary little in magnitude. For the fairly large n in Table 6.4 pairwise summation out-performs recursive summation (the insertion method is equivalent to pairwise summation in this example). The errors for compensated summation are zero for all the n we tried!

TABLE 6.4
x_i equispaced on $[1, 2]$.

n	Inc.	Dec.	Pair.	Comp.
2048	2.86e−6	3.86e−5	1.59e−7	0.00
	[2.80e6	3.50e6	3.38e4]	
	(2.40e1	3.24e2	1.33e0	0.00)
4096	3.35e−5	2.18e−5	1.59e−7	0.00
	[1.12e7	1.40e7	7.37e4]	
	(2.81e2	1.83e2	1.33e0	0.00)

In the next set of tests we used a MATLAB implementation [12] of the multidirectional search (MDS) method [39], [40] which attempts to locate a maximizer of $f : \mathbb{R}^n \to \mathbb{R}$ using function values only. We applied the maximizer to f defined as the relative error of the sum computed in single precision by recursive summation with the increasing ordering. With $n = 3$, starting with $x_0 = [\frac{1}{3}, \frac{2}{3}, 1]$, the maximizer located the following set of data after 280 function evaluations:

$$\begin{aligned} x &= [4.975987 \quad -2.495094 \quad -2.480894], \qquad f(x) = 1.0, \\ \widehat{S}_3 &= -9.5367 \times 10^{-7}, \qquad S_3 = -4.7684 \times 10^{-7}, \end{aligned}$$

where x and the S_3 values are quoted to seven and five significant figures, respectively. With f defined as the relative error for compensated summation, the MDS maximizer made little progress with the same starting value. But starting with $x_0 = [-\frac{1}{3}, 0, \frac{2}{3}]$, the maximizer found after 166 function evaluations the data

$$\begin{aligned} x &= [-0.8308306 \quad -0.7626623 \quad 1.593493], \qquad f(x) = 1.0, \\ \widehat{S}_3 &= 2.3842 \times 10^{-7}, \qquad S_3 = 1.1921 \times 10^{-7}. \end{aligned}$$

(If x is reordered with the increasing ordering then $f(x) = 1.0$, but $f(x) = 0$ for the decreasing ordering.)

These two examples are typical—using the maximizer it is straightforward to find data for which any of the summation methods yields no correct significant figures in the sum. The maximizer can also be used to compare two different methods, by defining f as the ratio of the errors from the two methods. With $n = 3$ we compared recursive summation (with the increasing ordering) with compensated summation. For both ratios of errors (E(Inc.)/E(Comp.) and its reciprocal) with certain starting values the maximizer was able to make the ratio arbitrarily large, by converging to data for which the error forming the denominator of f is zero. We observed similar behavior when comparing other methods.

Next, we describe an experiment with the forward substitution algorithm for solving a lower triangular system. We coded the inner product version of the algorithm and provided an option to choose between eight summation methods when evaluating the inner products. (The column-oriented form of forward substitution is not amenable to the use of different summation methods.) Lower triangular systems $Tx = b$ were solved in single precision and the forward error $\|\widehat{x} - x\|_\infty / \|x\|_\infty$ was evaluated for each of the eight summation options. We give results for $T = U^T$ where $PA = LU$ is the LU factorization with partial pivoting of the 10×10 Vandermonde matrix whose (i, j) element is $((j-1)/(n-1))^{i-1}$. In Table 6.5 we report results for the two systems with right-hand sides $b_i = Tx_i$, where x_i has elements equally spaced on the intervals $[1, 100]$ for $i = 1$ and $[0, 100]$ for $i = 2$. For this matrix $\kappa_\infty(T) = \|T\|_\infty \|T^{-1}\|_\infty \approx 3 \times 10^7$, and $\text{cond}(T, x_1) \approx \text{cond}(T, x_2) \approx 7 \times 10^5$, where $\text{cond}(T, x) = \| \, |T^{-1}||T||x| \, \|_\infty / \|x\|_\infty \le \kappa_\infty(T)$ is the condition number that appears in a forward error bound for the substitution algorithm [11]. The forward error varies over the different summation methods by a factor 98 for b_1 and a factor 39 for b_2; these are the largest variations we observed in tests with a variety of different matrices and right-hand sides. Throughout the tests there was no pattern to which summation method led to the smallest or largest forward error. This experiment shows that the choice of summation method for inner product evaluation can significantly affect the accuracy achieved by forward substitution, and this conclusion applies a fortiori to the solution of a full system via LU factorization. However, since there appears to be no straightforward way to predict which summation method will be the best for a given linear system, there is little reason to use anything other than recursive summation in the natural order when evaluating inner products within a general linear equation solver.

TABLE 6.5
Forward substitution with a 10×10 *matrix.*

	Orig.	Inc.	Dec.	Psum	Pair.	Ins.	+/−	Comp.
b_1	3.01e-4	1.18e-2	7.70e-3	1.18e-2	2.94e-2	1.18e-2	4.01e-3	7.70e-3
b_2	1.31e-2	2.64e-2	4.63e-3	2.64e-2	6.81e-4	1.06e-2	2.64e-2	2.04e-2

We have also experimented with compensated summation with the data arranged in order of decreasing magnitude. For all the problems we have tried, including those described above, the relative errors are $\le u_{SP}$. Our attempts to use the MDS maximizer to find a set of x_i for which the relative error exceeds u_{SP} have been unsuccessful. It is therefore natural to ask whether a relative error bound of the form $|E_n| \le cu|S_n|$ can be derived, where c is a constant independent of the x_i. The answer is no, because E_n can

can be nonzero when $S_n = 0$. Even when $n = 3$ and $S_n \neq 0$ it appears to be impossible to obtain such a bound. Nevertheless, our (limited) experience suggests that compensated summation with the decreasing ordering performs remarkably well in practice, and it would be interesting to determine why this is so.

Further test results can be found in the literature, although none are extensive. Linz [27] compares recursive summation with pairwise summation for uniform random numbers on $[0, 1]$ with $n = 2048$, averaging the errors over 20 trials, and Caprani [4] and Gregory [10] both conduct a similar experiment including compensated summation as well. Linnainmaa [26] applies recursive summation and compensated summation to series expansions, Simpson's rule for quadrature and Gill's Runge–Kutta method. Robertazzi and Schwartz [35] evaluate average mean square errors for recursive summation (with increasing, decreasing, and random orderings), pairwise summation and the insertion method, for the uniform $[0, 1]$ and exponential ($\mu = 0.5$) distributions with $n \leq 4096$.

7. Concluding remarks. No summation method from among those considered here can be regarded as superior to the rest from the point of view of accuracy, since for each method the error can vary greatly with the data, within the freedom afforded by the error bounds. However, some specific conclusions can be drawn.

1. For all but two of the methods the errors are, in the worst case, proportional to n. If n is very large, pairwise summation (error constant $\log_2 n$) and compensated summation (error constant of order 1) are attractive.

2. If the x_i all have the same sign, then all the methods yield a relative error of at most nu, and compensated summation guarantees perfect relative accuracy (as long as $nu \leq 1$). For recursive summation of one-signed data the increasing ordering is preferable to the decreasing ordering (and it is equivalent to the Psum ordering); however, the insertion method has the smallest bound (3.4) over all the methods considered here (excluding compensated summation).

3. For sums with heavy cancellation ($\sum_{i=1}^{n} |x_i| \gg |\sum_{i=1}^{n} x_i|$) recursive summation with the decreasing ordering is attractive (see Table 6.1), although it cannot be guaranteed to achieve the best accuracy (see Table 6.2).

Considerations of computational cost and the way in which the data are generated may rule out some of the methods. Recursive summation in the natural order, pairwise summation, and compensated summation can be implemented in $O(n)$ operations for general x_i, but the other methods are more expensive since they require searching or sorting. Furthermore, in an application such as the numerical solution of ordinary differential equations where the x_i are generated sequentially, and x_k may depend on $\sum_{i=1}^{k-1} x_i$, sorting and searching may be impossible. One way to achieve higher accuracy that we have not mentioned is simply to implement recursive summation in higher precision; if this is feasible, it may be less expensive (and more accurate) than using one of the alternative methods in working precision.

Finally, we return to the two practical applications mentioned in the introduction. In [24], the $+/-$ method was found to cure some problems with inaccurate gradients in an optimization method. This is a little surprising since we have found the $+/-$ method to be unattractive. It appears that there is some feature of this application, not apparent from [24], that encourages the $+/-$ method to perform better than recursive summation with the natural ordering. The loss of symmetry in a quasi-Newton method that was observed in [7] is easier to understand. For example, symmetries in F in (1.1) can be preserved by using any summation method whose computed answer does not depend on the given order of the data—such as recursive summation with the increasing ordering and with elements of equal magnitude ordered by sign.

Acknowledgments. I thank Jeremy Du Croz and Philip Gill for providing some of the references, and Des Higham, Nick Trefethen, and the referee for suggesting improvements to the manuscript.

REFERENCES

[1] I. Babuška, *Numerical stability in mathematical analysis*, in Proc. IFIP Congress, Information Processing 68, North-Holland, Amsterdam, 1969, pp. 11–23.
[2] ———, *Numerical stability in problems of linear algebra*, SIAM J. Numer. Anal., 9 (1972), pp. 53–77.
[3] G. Bohlender, *Floating-point computation of functions with maximum accuracy*, IEEE Trans. Comput., C-26 (1977), pp. 621–632.
[4] O. Caprani, *Implementation of a low round-off summation method*, BIT, 11 (1971), pp. 271–275.
[5] T. J. Dekker, *A floating-point technique for extending the available precision*, Numer. Math., 18 (1971), pp. 224–242.
[6] J. W. Demmel, *Underflow and the reliability of numerical software*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 887–919.
[7] L. C. W. Dixon and D. J. Mills, *The effect of rounding error on the variable metric method*, Tech. Rep. 229, Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, UK, Apr. 1990.
[8] S. Gill, *A process for the step-by-step integration of differential equations in an automatic digital computing machine*, Proc. Cambridge Phil. Soc., 47 (1951), pp. 96–108.
[9] D. Goldberg, *What every computer scientist should know about floating-point arithmetic*, ACM Comput. Surveys, 23 (1991), pp. 5–48.
[10] J. Gregory, *A comparison of floating point summation methods*, Comm. ACM, 15 (1972), p. 838.
[11] N. J. Higham, *The accuracy of solutions to triangular systems*, SIAM J. Numer. Anal., 26 (1989), pp. 1252–1265.
[12] ———, *Optimization by direct search in matrix computations*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 317–333.
[13] R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger, Bristol, 1981.
[14] M. Jankowski, A. Smoktunowicz, and H. Woźniakowski, *A note on floating-point summation of very many terms*, J. Inform. Process. Cybern., 19 (1983), pp. 435–440.
[15] M. Jankowski and H. Woźniakowski, *The accurate solution of certain continuous problems using only single precision arithmetic*, BIT, 25 (1985), pp. 635–651.
[16] W. Kahan, *Further remarks on reducing truncation errors*, Comm. ACM, 8 (1965), p. 40.
[17] ———, *A survey of error analysis*, in Proc. IFIP Congress, Ljubljana, Information Processing 71, North-Holland, Amsterdam, 1972, pp. 1214–1239.
[18] ———, *Implementation of algorithms* (*lecture notes by W. S. Haugeland and D. Hough*), Tech. Rep. 20, Dept. of Computer Science, Univ. of California, Berkeley, 1973.
[19] ———, *Doubled-precision* IEEE *standard* 754 *floating-point arithmetic*, manuscript, Feb. 1987.
[20] ———, *How Cray's arithmetic hurts scientific computation* (*and what might be done about it*), manuscript prepared for the Cray User Group meeting in Toronto, June 1990.
[21] A. Kielbasiński, *Summation algorithm with corrections and some of its applications*, Math. Stos., 1 (1973), pp. 22–41. (In Polish, cited in [14] and [15].)
[22] D. E. Knuth, *The Art of Computer Programming, Volume* 2, *Seminumerical Algorithms*, 2nd Ed., Addison-Wesley, Reading, MA, 1981.
[23] U. W. Kulisch and W. L. Miranker, *The arithmetic of the digital computer: A new approach*, SIAM Rev., 28 (1984), pp. 1–40.
[24] L. S. Lasdon, J. Plummer, B. Buehler, and A. D. Waren, *Optimal design of efficient acoustic antenna arrays*, Math. Prog., 39 (1987), pp. 131–155.
[25] H. Leurecht and W. Oberaigner, *Parallel algorithms for the rounding exact summation of floating point numbers*, Computing, 28 (1982), pp. 89–104.
[26] S. Linnainmaa, *Analysis of some known methods of improving the accuracy of floating-point sums*, BIT, 14 (1974), pp. 167–202.
[27] P. Linz, *Accurate floating-point summation*, Comm. ACM, 13 (1970), pp. 361–362.
[28] M. A. Malcolm, *On accurate floating-point summation*, Comm. ACM, 14 (1971), pp. 731–736.
[29] D. D. McCracken and W. S. Dorn, *Numerical Methods and Fortran Programming*, John Wiley, New York, 1964.
[30] C. B. Moler, J. N. Little, and S. Bangert, *PC-Matlab User's Guide*, The MathWorks, Inc., Natick, MA, 1987.

[31] O. MØLLER, *Quasi double-precision in floating point addition*, BIT, 5 (1965), pp. 37–50.

[32] A. NEUMAIER, *Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen*, Z. Angew. Math. Mech., 54 (1974), pp. 39–51.

[33] K. NICKEL, *Das Kahan-Babuškasche Summierungsverfahren in Triplex-ALGOL* 60, Z. Angew. Math. Mech., 50 (1970), pp. 369–373.

[34] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[35] T. G. ROBERTAZZI AND S. C. SCHWARTZ, *Best "ordering" for floating-point addition*, ACM Trans. Math. Software, 14 (1988), pp. 101–110.

[36] D. R. ROSS, *Reducing truncation errors using cascading accumulators*, Comm. ACM, 8 (1965), pp. 32–33.

[37] I. A. STEGUN AND M. ABRAMOWITZ, *Pitfalls in computation*, J. Soc. Indust. Appl. Math., 4 (1956), pp. 207–219.

[38] F. STUMMEL, *Rounding error analysis of elementary numerical algorithms*, Computing, Suppl., 2 (1980), pp. 169–195.

[39] V. J. TORCZON, *Multi-directional Search: A direct search algorithm for parallel machines*, Ph.D. thesis, Rice University, Houston, TX, May 1989.

[40] ———, *On the convergence of the multi-directional search algorithm*, SIAM J. Optimization, 1 (1991), pp. 123–145.

[41] E. VITASEK, *The numerical stability in solution of differential equations*, in Conference on the Numerical Solution of Differential Equations, Lecture Notes in Mathematics 109, Springer-Verlag, Berlin, 1969, pp. 87–111.

[42] J. H. WILKINSON, *Error analysis of floating-point computation*, Numer. Math., 2 (1960), pp. 319–340.

[43] ———, *Rounding Errors in Algebraic Processes*, Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963.

[44] J. M. WOLFE, *Reducing truncation errors by programming*, Comm. ACM, 7 (1964), pp. 355–356.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 800–823, July 1993

ADAPTIVE LINEAR EQUATION SOLVERS IN CODES FOR LARGE STIFF SYSTEMS OF ODES*

K. R. JACKSON[†] AND W. L. SEWARD[‡]

Abstract. Iterative linear equation solvers have been shown to be effective in codes for large systems of stiff initial-value problems for ordinary differential equations (ODEs). While preconditioned iterative methods are required in general for efficiency and robustness, unpreconditioned methods may be cheaper over some ranges of the interval of integration. In this paper, a strategy is developed for switching between unpreconditioned and preconditioned iterative methods depending on the amount of work being done in the iterative solver and properties of the system being solved. This strategy is combined with a "type-insensitive" approach to the choice of formula used in the ODE code to develop a method that makes a smooth transition between nonstiff and stiff regimes in the interval of integration. As expected, it is found that for some large systems of ODEs, there may be a considerable saving in execution time when the type-insensitive approach is used. If there is a region of the integration that is "mildly" stiff, switching between unpreconditioned and preconditioned iterative methods also increases the efficiency of the code significantly.

Key words. type-insensitive ODE code, iterative linear solver, preconditioning

AMS subject classifications. 65L05, 65F10

1. Introduction.

In recent years, there have been several investigations of the use of iterative linear equation solvers in codes for the numerical solution of large systems of stiff initial-value problems (IVPs) for ordinary differential equations (ODEs). See, for example, [5], [6], [8], and [12]. Such work has established clearly the potential effectiveness of the combination of these methods, but there are many open questions concerning both the choice of iterative method and the way in which it interacts with strategies used in the ODE solver. Frequently, the iterative methods investigated have been of the "Krylov subspace" type, e.g., the conjugate gradient method, Orthomin [22], or GMRES [6]. In this paper, we also consider an iterative method of this type, but our results should apply to a broader class since we are primarily concerned with the interaction between the iterative method and the other strategies of the ODE solver.

We study the numerical solution of large systems of stiff IVPs for ODEs of the form

$$y' = f(t, y), \qquad y(t_0) \text{ given.} \tag{1}$$

Due to the stiffness, such problems are usually discretized using an implicit numerical method, most often the backward differentiation formulas (BDFs) [11]. We will concentrate on the BDFs, although many of our ideas apply to implicit numerical methods in general. Applied to (1), a k-step BDF has the form

$$y_n = \sum_{j=1}^{k} \alpha_n^j y_{n-j} + h_n \beta_n f(t_n, y_n). \tag{2}$$

To find y_n from (2), it is necessary to solve a system of equations of the form

$$F(y_n) = y_n - h_n \beta_n f(t_n, y_n) - \phi_n = 0, \tag{3}$$

*Received by the editors August 5, 1991; accepted for publication (in revised form) July 15, 1992. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and by the Information Technology Research Centre of Ontario.

[†]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.
[‡]Algorithmics Inc., 822 Richmond Street West, Toronto, Ontario, Canada M6J 1C9.

where ϕ_n contains the terms in (2) that do not depend on y_n. In general, this system is nonlinear and is solved by Newton's method or some variant of it.

The usual form of Newton's method is

```
• select an initial guess $y_n^0$
• for $l = 0, 1, \ldots$
    —solve $F_y(y_n^l)\Delta_n^l + F(y_n^l) = 0$ for $\Delta_n^l$,
    —set $y_n^{l+1} = y_n^l + \Delta_n^l$,
```

where $F_y(y_n^l) = I - h_n\beta_n f_y(t_n, y_n^l)$. Most ODE codes use a chord method (often called a pseudo- or simplified-Newton method) in which $F_y(y_n^l)$ is replaced by some approximation W_n^l, generally the Newton iteration matrix F_y (or some approximation to it) from some earlier timestep. If the system of ODEs (1) is large, the linear algebra cost of the *solve* step is normally a major part of the cost of the integration. If W_n^l is sparse, as is usually the case for large systems, exploiting the sparsity can be an effective means of reducing this cost. Using a direct sparse matrix solver in Newton's method is reasonably straightforward and has been discussed in the context of ODE solvers by, for example, Berzins and Furzeland [3] and Sherman and Hindmarsh [19]. Chan and Jackson [8] compare direct and iterative methods in this context. In this paper, we concentrate on the use of iterative solvers.

When an iterative linear equation solver is used in the *solve* phase of Newton's method, an *inexact Newton method* [10] is obtained. In general, such a method has the form

```
• select an initial guess $y_n^0$
• for $l = 0, 1, \ldots$
    —find $\Delta_n^l$ satisfying $F_y(y_n^l)\Delta_n^l + F(y_n^l) = d_n^l$ for $d_n^l$ "sufficiently small,"
    —set $y_n^{l+1} = y_n^l + \Delta_n^l$.
```

This iteration converges at least linearly if

$$\|d_n^l\| \le \eta_l \|F(y_n^l)\|, \tag{4}$$

where $0 \le \eta_l \le \eta < 1$. In the case where an iterative linear equation solver is used to find y_n^l, the residual d_n^l is the final residual of the linear iteration.

Brown and Hindmarsh [5] consider iterative linear equation solvers based on Arnoldi's method for use in the ODE code LSODE [14]. In their approach, the matrix-vector product required in the iterative method is approximated by a finite difference based on f. This approach avoids explicitly forming the iteration matrix W_n^l needed in Newton's method and hence has been referred to as a "matrix-free" method. These methods are very efficient in storage and, as predicted by the theory of Krylov subspace methods [5], converge quickly if the eigenvalues of W_n^l are clustered. If this is not the case, preconditioning is usually required for efficiency and robustness in the ODE solver. In [6], preconditioning strategies for use with the "matrix-free" approach are discussed. These strategies require formation and storage of at least part of W_n^l in general, but are still very efficient in terms of storage. The authors find that, with preconditioning, a wider class of problems can be solved efficiently.

Chan and Jackson [8] also investigate both unpreconditioned and preconditioned Krylov-type iterative methods used with LSODE. The methods they use are not "matrix-free"—W_n^l is formed, stored and used explicitly to compute the matrix-vector products needed in the iterative method. The authors consider various ways of reducing the amount of time spent in forming and processing W_n^l. For example, if preconditioning is not being used, they show that it is not difficult to keep the factor $h_n\beta_n$ in W_n^l current, thus requiring fewer Jacobian evaluations for the integration.

Hence we observe that, for a variety of reasons, unpreconditioned iterative methods can be cheaper to use than preconditioned ones in stiff ODE solvers. However, preconditioning appears necessary in general for efficiency and robustness. This observation led us to study an adaptive approach to switching preconditioning on and off. In particular, if the eigenvalues of the Newton iteration matrix W_n^l are clustered, which always occurs when h_n is small, preconditioning may not be necessary. In other cases, it is likely that the number of iterations of the linear equation solver can be reduced significantly by preconditioning.

Instead of not preconditioning at all, our method uses *diagonal scaling*, a cheap preconditioner described in more detail below, when this seems to be effective, but switches to a more powerful preconditioner based on W_n^l when diagonal scaling appears to be ineffective. We expect that CPU time can be saved by avoiding unnecessary expensive preconditioning without sacrificing robustness since preconditioning is available when necessary.

The idea of switching preconditioning on and off combines naturally with the use of a "type-insensitive" ODE solver. A type-insensitive code switches between methods appropriate for nonstiff and stiff IVPs depending on the nature of the problem. This approach has been discussed by several authors for a variety of formulas. See, for example, [4], [15]–[18]. Petzold [16] developed a type-insensitive method based on the Adams formulas [11] and the BDFs that switches between the two classes of formulas based on Jacobian information and behaviour of the code. We have combined the ideas from [16] with our adaptive approach to preconditioning. Our code starts integrating with the Adams formulas and switches to the BDFs as described by Petzold. When this switch is made, the linear equation solver uses diagonal scaling only. Later the method may switch to a more expensive preconditioner described below. It can switch back to diagonal scaling and to the Adams formulas, if it predicts they will be more cost effective.

The basic iterative linear equation solver that we use is Orthomin [22]. Other reasonable choices include GMRES [5] and the stabilized version of the conjugate-gradient-squared (CGS) algorithm [21]. To focus on adaptive preconditioning, we chose to use one basic iterative solver only. We selected the well-known scheme Orthomin, in part because an effective implementation of it along with several preconditioners was readily available to us.

There is a variety of possible approaches for the "cheap" preconditioning. We could use no preconditioning at all, in which case a "matrix-free" method or an approach that updates $h_n\beta_n$ frequently might be very efficient. This approach would allow us to avoid forming W_n^l entirely or to form it infrequently during this phase. However, there may be hidden costs associated with such methods. For example, a "matrix-free" method requires one function evaluation per linear iteration. Forming the Newton iteration matrix W_n^l explicitly, on the other hand, is frequently a relatively inexpensive operation for a large sparse system of ODEs, costing a few function evaluations only.

In our numerical experiments, we chose to form W_n^l explicitly, according to the heuristics of the basic ODE solver, and to use diagonal scaling (that is, preconditioning by a diagonal matrix to make the diagonal elements of W_n^l all ones) as our cheap preconditioner. The improvement in performance when diagonal scaling is used can be significant even though the cost is low. Moreover, diagonal scaling often reduces the cost of the iterative solver since it is not necessary to multiply by the matrix diagonal in this case when computing matrix-vector products. So there is little point to omitting diagonal scaling if W_n^l is computed explicitly. Furthermore, diagonal scaling can be used with other approaches to forming W_n^l (see [5], for example, for a discussion of scaling with "matrix-free" methods), but we have not attempted any elaboration of our basic approach. For brevity, in the remainder of the paper, we occasionally refer to an iterative method with diagonal scaling only as an unpreconditioned method.

A more powerful preconditioner is switched on if diagonal scaling appears to be ineffective. The particular preconditioning strategies that we have studied are Level 0 and Level 1 incomplete LU (ILU) factorizations, as described in §2. We note that diagonal scaling is also used with the ILU preconditioners, as it is often found to improve their performance. These preconditioning strategies can be applied to any matrix in a "black box" fashion, and have been found to work well in solving time-dependent partial differential equations (PDEs). (See, for example, [1], [2], and [9].) While these preconditioners depend on the explicit presence of the Jacobian matrix, the idea of switching between preconditioners should be useful whenever a variety of strategies of differing costs are available.

We also develop criteria for switching these preconditioners off, and this is the only place in our adaptive strategy where we use Jacobian information directly. Our strategy looks at both the performance of the iterative solver and the spread of the eigenvalues of the Newton iteration matrix W_n^l. The eigenvalue information is useful since fast convergence of the iterative solver might be primarily due to the preconditioner. We have not investigated the case in which the Jacobian is not available explicitly. In most cases, the development of a good preconditioner requires at least some Jacobian information which could be used in our test, possibly in a more conservative form.

In the next section, we briefly describe the codes that were used in this investigation. Heuristics for identifying when to switch preconditioning on and off are discussed in §3. The use of adaptive preconditioning in type-insensitive codes is considered in §4. Test problems and numerical results are presented in both §3 and §4 to illustrate the performance of the techniques. We end with some conclusions in §5.

2. Codes used in the investigation. The ODE package used in this investigation was SPRINT [3]. This package is designed to offer the user a range of methods for the numerical solution of systems of IVPs in ODEs. It allows the user to select from four timestepping methods and three linear algebra packages to create the complete method appropriate for a particular problem. The three algebra routines are full, banded, and sparse direct solvers. Because of this modularity, the SPRINT package is well suited for testing the interaction of the iterative linear algebra solver with the ODE method, since it is possible to couple our iterative solver to a sophisticated, fully developed timestepping scheme.

SPRINT includes a space discretization module, but for one-dimensional PDE systems only. Consequently, the higher-dimensional PDE test problems used in this paper have been semidiscretized in space by our own routines. One motivation for our study of iterative linear solvers is the current work being done with SPRINT to develop modules for spatial discretization of PDE systems in higher dimensions.

The particular ODE method used in this investigation is based on the BDFs and is similar in implementation to the well-known code LSODE [14]. A variant of LSODE, called LSODA, was written by Hindmarsh and Petzold to incorporate the ideas for a type-insensitive method described in [16]. Our type-insensitive ODE solver was developed by altering the SPRINT BDFs module corresponding to the changes made to evolve LSODE into LSODA.

The iterative solver used in our study, WATSIT, was developed at the University of Waterloo based on methods described in [1], [2], and [9]. This code is designed to solve systems of linear equations of the form $Ax = b$ where the matrix A is large and sparse. The basic technique is an incomplete factorization scheme with acceleration. The user has several choices for both the acceleration method and the incomplete factorization. Possible acceleration methods include the conjugate gradient method, Orthomin [22], the CGS algorithm [20], and a stabilized version of CGS [21]. As noted in §1, since we wish to focus on our adaptive strategy in this paper, we use Orthomin acceleration only in our numerical tests and have not encountered any test problems where it breaks down.

The user has the choice of diagonal scaling only or preconditioning based on either Level 0 or Level 1 ILU factorization, denoted ILU(0) and ILU(1), respectively. These preconditioners are defined by discarding all or some of the fill elements that would arise during a complete LU decomposition of the matrix associated with the system to be solved. ILU(0), the simpler of the two preconditioners, discards all fill elements. Thus, this preconditioner has the same sparsity structure as the original matrix, which can be an advantage since the storage requirement is known a priori. The more powerful ILU(1) preconditioner retains the fill generated by eliminating the original nonzero elements. These and other more powerful preconditioners may reduce the number of iterations required to solve a linear system, although each iteration with a more powerful preconditioner is generally more expensive than one with diagonal scaling or no preconditioner at all. Whether a more powerful, but more expensive, preconditioner will be more cost effective than a weaker, but less expensive, one depends on the linear system to be solved. Further discussion of ILU preconditioners can be found in [9] and [13].

In Brown and Hindmarsh [5], convergence criteria for the iterative method are discussed in detail in the context of an ODE code. Both Brown and Hindmarsh [5] and Chan and Jackson [8] find that the residual reduction condition (4) is overly expensive: it forces linear iterations that do not seem to improve the accuracy of the ODE solution significantly. In [5], the authors propose and justify a strategy that accepts the result produced by the iterative linear solver if the residual is some small fraction of the tolerance level required on the solution of the nonlinear system. A similar strategy was developed in [8]. We adopt this approach and also use the same fraction that is suggested in [5], namely, $1/20$. The residual is measured in the weighted norm used by the ODE solver. Since the acceleration methods used in WATSIT return the residual directly, we do not need to scale the matrix to obtain the weighted norm, as is done in [5]. The linear iteration is started with a zero vector as its initial guess and always does at least one iteration. If the maximum iteration count is reached without the desired reduction of the residual on the first Newton iteration, the code accepts Δ_n^0 and updates $y_n^1 = y_n^0 + \Delta_n^0$ provided that the current residual, $F_y(y_n^0)\Delta_n^0 + F(y_n^0) = d_n^0$, is smaller than the initial one, $F(y_n^0)$. Otherwise, a failure is signalled to the ODE solver and either the Newton iteration or the timestep is retried, according to the criteria in the ODE code for a failure of Newton's method.

All numerical results reported in this paper were computed in double precision on an MIPS M/120 RisComputer.

3. Adaptive preconditioning.

In this section, the adaptive choice of preconditioning method is discussed independently of the type-insensitive approach for solving stiff ODEs. The two ideas are combined in §4. To develop strategies for switching between preconditioning methods, we need to assess the relative cost of the preconditioners as well as the expected saving when the switch is made.

For a particular iterative method and implementation, it is straightforward to estimate the relative cost of preconditioning by counting floating-point operations in the code. For simplicity, we estimate this operation count based on the number of elements per row of the matrix. If the original matrix has on average *nrow* nonzero elements per row and the preconditioned matrix has on average *pnrow* elements, then, for our implementation of Orthomin, the following table gives the costs for a matrix of dimension *neq*, where the cost of diagonal scaling is included in all cases.

Operation counts for the iterative methods.

Scheme	Initialization	k iterations
No pre	$(2\,nrow + 4)\;neq$	$(2\,nrow + 11)\;neq\;k + 3\;neq\;k\,(k-1)$
Pre	$(2\,pnrow + 4)\;neq$	$(4\,pnrow + 12)\;neq\;k + 3\;neq\;k\,(k-1)$

For example, for the heat equation $u_t = u_{xx} + u_{yy} + u_{zz}$ in three space dimensions and a centered finite difference spatial discretization, $nrow = 7$; for ILU(0), $pnrow = 7$; and, for ILU(1), $pnrow = 13$. Using these values, we can calculate the relative costs for different iteration counts shown in Table 1. We note that the iterations with ILU(0) preconditioning are only about 40% more expensive than those that use only diagonal scaling. Of course, the ILU-preconditioned iterations incur the extra cost of doing the ILU factorization, $O(pnrow^2 \cdot neq)$ floating-point operations. This cost is roughly that of doing a few linear iterations. Since we are using a chord method rather than full Newton iteration, and the factorization is only done when found necessary by the ODE solver, this cost is small compared to the total cost of linear iterations. We do not include here the cost of forming the matrix W_n^l, since this is done in all cases. If the cheap preconditioning strategy also avoided the formation of W_n^l, then forming W_n^l would be an additional cost of using a more powerful preconditioner.

TABLE 1
Relative cost of iterative methods applied to the heat equation.

Iterations	Scaling	ILU(0)	ILU(1)
1	43 *neq*	58 *neq*	94 *neq*
2	74 *neq*	104 *neq*	164 *neq*
3	111 *neq*	156 *neq*	240 *neq*
4	154 *neq*	204 *neq*	322 *neq*
5	203 *neq*	278 *neq*	410 *neq*

In one single solve of a linear system, a preconditioned method will be cheaper than an unpreconditioned one if the number of linear iterations is reduced sufficiently by preconditioning. The necessary reduction can be shown clearly, as in the tables above. Over the course of the integration of an ODE, the trade-off is more involved. The basic strategy for a general-purpose ODE solver must be to use a preconditioned iterative method for robustness. By substituting an unpreconditioned iteration in some cases, we hope to reduce the total cost of linear algebra, but we expect to see the total number of linear iterations increase, as a few expensive iterations are replaced by a larger number of cheaper ones.

We address the question of switching from diagonal scaling to a more powerful preconditioner first. Diagonal scaling is used at the start of the integration, since most ODE codes begin timestepping with a small stepsize, and it is usual to find an initial region of the integration where the stepsize remains small due to the accuracy requirement. The progress of the integration is then monitored with the aim of switching to a more powerful preconditioner when diagonal scaling appears to require too many iterations. An obvious strategy for assessing when the diagonally scaled method is using "too many" iterations is to wait until the iteration fails to meet the convergence criterion in the maximum number of iterations. Frequently, this maximum number is set fairly small. For example, Brown and Hindmarsh [5] found that five iterations worked well. In this case, switching preconditioning on when the diagonally scaled iteration "fails" is not unreasonable. In our tests, we have used a maximum value of 10 iterations to try to assess the switching strategy independently of convergence failures. We have found that the methods are more efficient if preconditioning is switched on when the diagonally scaled method is taking only a few (e.g., two to four) iterations per solve.

It turns out to be straightforward to assess the number of linear iterations that the method uses per solve. We have observed in practice that the number of linear iterations per Newton iteration tends to remain constant over several Newton iterations. That is, typically only one linear iteration per Newton iteration is required at the start of an integration. This number increases to two at some point as the stepsize increases, remains steady at two for a while, then increases to three, and so on. Consequently, it is possible to estimate the number of linear iterations per solve by taking the average over the last few Newton iterations.

We have not attempted to use any theoretical results to predict the reduction in the number of linear iterations when preconditioning is applied. Such results are available for some problems, preconditioners, and iterative methods, but they give upper bounds on, or asymptotic estimates of, the number of iterations required to reduce the residual to some fraction of its initial size. Since our convergence criterion, discussed in §2, only requires reducing the residual to a fixed value and since we expect to use only a few iterations per solve, the theoretical results are not particularly useful here.

When a differential equation yields linear systems with clustered eigenvalues, unpreconditioned and diagonally scaled iterative methods can be very effective. In [5], the authors use a reaction-diffusion system with two species in three space dimensions as a test problem. The differential equations have the form

$$\begin{gathered}
\frac{\partial c^i}{\partial t} = d_i \Delta c^i + f^i(c^1, c^2), \qquad i = 1, 2, \\
d_1 = 0.05, \qquad d_2 = 1.0, \\
f^1(c^1,c^2) = c^1(b_1 - a_{11}c^1 - a_{12}c^2), \qquad f^2(c^1,c^2) = c^2(b_2 - a_{21}c^1 - a_{22}c^2), \\
a_{11} = 10^6, \quad a_{12} = 1, \quad a_{21} = 10^6 - 1, \quad a_{22} = 10^6, \\
b_1 = b_2 = (1 + \alpha xyz)(10^6 - 1 + 10^{-6}).
\end{gathered} \tag{5}$$

The equations are defined on the unit cube with $t \in [0, 10]$, homogeneous Neumann boundary conditions and initial conditions

$$\begin{aligned}
c^1(x,y,z,0) &= 500 + 250\cos(\pi x)\cos(3\pi y)\cos(10\pi z), \\
c^2(x,y,z,0) &= 200 + 150\cos(10\pi x)\cos(\pi y)\cos(3\pi z).
\end{aligned}$$

The authors point out that, as $t \to \infty$, the solution approaches a steady state which is given roughly by the asymptotic solution of the problem without diffusion, namely,

$$c^1 = (1 - 10^{-6})(1 + \alpha xyz), \qquad c^2 = 10^{-6}(1 + \alpha xyz).$$

The spatial derivatives are discretized using a centered second-order finite difference approximation on an evenly spaced mesh with m subdivisions in each direction, yielding a system of $2(m+1)^3$ ODEs. Near the steady state, the interaction terms dominate the Jacobian and the dominant part of the spectrum is clustered in the interval -10^6 to $-10^6(1+\alpha)$. Brown and Hindmarsh [5] point out that, consequently, unpreconditioned iterative methods are expected to work well unless α is large. It does not necessarily follow that diagonal scaling will work well, but in this case it does, since all eigenvalues of the scaled matrix are of moderate size. In [5], the ODE problem is solved using a relative error tolerance of 10^{-6} and an absolute error tolerance of 10^{-8}. We use the same tolerances here, with the result that the code takes a very small stepsize (starting around 10^{-9}) through the transient region, which lasts to about $t = 0.5$. There is then an abrupt transition to the steady-state region, during which the stepsize increases by factors of two to five several times in rapid succession. The diagonally scaled method encounters difficulty once the transition to the steady-state region starts. Numerical results are shown in Tables 2 and 3 for the case $m = 9$ (2000 equations) and for $\alpha = 0$. The maximum number of linear iterations was set to 10.

TABLE 2
Reaction-diffusion equation in three dimensions; no switching.

	Diagonal	ILU(0)	ILU(1)
NFCN	1490	1466	1486
NJAC	71	67	68
NNWT	617	641	649
NLIN	1005	737	712
TIME	320	322	352

TABLE 3
Reaction-diffusion equation in three dimensions; switching.

	ILU(0)				ILU(1)			
	2	4	6	MAXIT	2	4	6	MAXIT
NFCN	1402	1390	1402	1417	1403	1404	1416	1416
NJAC	65	64	65	66	65	65	66	66
NNWT	601	601	601	604	602	603	603	603
NLIN	733	773	796	848	679	743	773	814
TSWI	217	243	253	263	217	243	256	266
TIME	278	280	284	295	280	281	288	293

Table 2 shows the number of function evaluations (NFCN), Jacobian evaluations (NJAC), Newton iterations (NNWT), linear iterations (NLIN), and the time in seconds (TIME) required to integrate the problem without adaptation of the preconditioning, using either diagonal, ILU(0), or ILU(1) preconditioning. Table 3 shows the effect of switching from diagonal scaling to ILU(0) or ILU(1) preconditioning during the integration. Each column corresponds to the number of linear iterations being used by the diagonally scaled method when the switch was made. This number was calculated by looking at the average number of iterations over the last four Newton iterations. The column labelled MAXIT indicates that the switch was made because the diagonally scaled method failed to converge in the maximum number of iterations. The criterion used in the program was to switch if the code took an average of eight linear iterations over the past four Newton iterations, but a convergence failure occurred in each case before this condition was met. The row of the table labelled TSWI shows the elapsed CPU time when the code switched on the preconditioning.

From these tables, we see that there is a definite advantage to using the combination of the diagonally scaled and preconditioned iterative methods, as most of the CPU time is spent with the diagonally scaled iteration. This reflects the fact that 75 to 90% of the work of the integration, measured in function evaluations, Jacobian evaluations, or nonlinear or linear iterations is done using that diagonally scaled method. There is little to choose between switching criteria in terms of total time, suggesting that there is a small range of the integration in which the diagonally scaled and preconditioned methods are equally expensive.

In the following tables, we show some numerical results obtained using the heat equation in three space dimensions as a test problem. It is well known that the eigenvalues of the associated Jacobian matrix are widely spread without clustering and that unpreconditioned and diagonally scaled iterative methods are generally inefficient. As described in [8], the PDE has the form

$$u_t = u_{xx} + u_{yy} + u_{zz} \tag{6}$$

on the unit square for $t \in [0, 10.24]$, with homogeneous Dirichlet boundary conditions and initial condition

$$u(0, x, y, z) = 64x(1-x)y(1-y)z(1-z).$$

The spatial derivatives were discretized using a centered second-order finite difference discretization on a mesh of m equal subdivisions, resulting in a system of $(m-1)^3$ ODEs. In this example, $m = 16$ (3375 equations). The maximum number of linear iterations was set to 10. The absolute error tolerance in the ODE solver was set to 10^{-4} and no relative error control was used. Table 4 shows the results without adaptation of the preconditioner, and Table 5 shows the effect of switching preconditioners. The number of linear iterations was calculated as the average over the last four Newton iterations.

TABLE 4
Heat equation; no switching.

	Diagonal	ILU(0)	ILU(1)
NFCN	356	373	326
NJAC	19	20	17
NNWT	117	122	113
NLIN	400	200	140
TIME	130	107	94

The most efficient way to solve this particular problem is to use the ILU(1) preconditioner over all iterations. This seems to be due, at least in part, to accuracy considerations related to the use of the diagonally scaled iterative solver. Following Brown and

TABLE 5
Heat equation; switching.

	ILU(0)				ILU(1)			
	2	4	6	MAXIT	2	4	6	MAXIT
NFCN	357	358	369	372	373	360	369	373
NJAC	19	19	20	20	20	19	20	20
NNWT	118	119	118	121	122	121	118	122
NLIN	189	244	320	359	170	212	306	340
TSWI	22	48	90	100	22	48	90	100
TIME	98	106	118	127	102	103	117	125

Hindmarsh [5], the solution of the linear system is accepted if

$$\|d_n^l\| \leq \epsilon,$$

where d_n^l is the residual at the last linear iteration. With two different iterative methods, it often happens that one method just meets this criterion while the other returns a solution with a much smaller residual. This difference in accuracy affects not only the acceptance of the Newton iteration but also the error control strategy and stepsize selection in the ODE solver. In the tables, we note that, as expected, the total number of linear iterations increases as we use the diagonally scaled method for a longer period. The other statistics are more variable because of this effect of the accuracy of the linear solver.

For the ILU(0) preconditioning, the approaches that switch when the diagonally scaled method is taking two or four iterations per solve are no worse than using preconditioning on all steps. Switching always leads to a deterioration in performance for the ILU(1) preconditioner.

For both the reaction-diffusion problem (5) and the heat equation (6), we have tested the effect of averaging the number of linear iterations over the last six and the last eight Newton iterations. Changing this parameter has little effect on the step at which preconditioning is switched on.

Based on the results from these two test problems and additional numerical experiments, it seems reasonable to use the following strategy to switch from a diagonally scaled method to a preconditioned one:

- Find the average number of linear iterations over the past four Newton iterations.
- Switch on preconditioning when the average is four or greater, and definitely when the diagonally scaled iteration fails to converge in the maximum number of iterations.

This approach sometimes yields a significant improvement in performance; for those problems where it is less efficient, it does not degrade performance very much. The particular value of *four* linear iterations obviously depends on the relative costs of the preconditioning strategies and hence is dependent on the problem class, but does seem appropriate for the PDE problems that we have tested.

For either the reaction-diffusion equation (5) or the heat equation (6), we observe that the number of linear iterations per solve is reduced when preconditioning is first switched on. As the integration continues with the preconditioned method, the number of iterations per solve increases again, along with the timestep, as the solution of the differential equation approaches a steady state. This is typical behaviour for many time-dependent systems, but problems exist for which phases similar to the initial transient recur. For such problems, it is reasonable to consider switching preconditioning off. We have developed a test to switch preconditioning off based on both the amount of work being done by the linear solver and the character of the iteration matrix. As the primary test, we require that the iterative solver takes one iteration only per solve over the last several Newton iterations. Around 15 to 20 Newton iterations seems appropriate since we wish to avoid cases where, say, the error estimate has forced an anomalously small stepsize over a few steps. This also avoids switching preconditioning off again immediately after switching it on.

A quantity relevant to convergence of a Krylov subspace-type iterative method is the *spectral ratio*—the ratio of the largest to the smallest eigenvalues of the matrix or, for clustered eigenvalues, the ratio of largest to smallest eigenvalues within each cluster. We use Gerschgorin circles to estimate the clustering of the eigenvalues of the Newton iteration matrix. We need to consider one cluster only since all the Gerschgorin circles of the diagonally scaled matrix will have center $(1,0)$. We also observe that the matrices $D^{-1}W_n^l$, $W_n^l D^{-1}$, and $D^{-1/2}W_n^l D^{-1/2}$ are similar; hence they have the same eigenvalues, but need not have the same Gerschgorin circles. ($D = \operatorname{diag}(W_n^l)$ is the diagonal scaling matrix.) When the Newton iteration matrix is formed by the ODE solver, the Gerschgorin circles for these three matrices are calculated by both rows and columns, giving six estimates of the eigenvalue range. If all the circles for a single case (particular scaling and row or column calculation) lie in the same half-plane, we call the ratio of the extreme real axis intercepts of the Gerschgorin circles a *Gerschgorin ratio* of the matrix. We take the minimum of our six calculated Gerschgorin ratios and use it to approximate the associated spectral ratio. When this value is small, it is reasonable to expect that an unpreconditioned iterative method will work well.

As the theory suggests, we have found for all our test problems that, at the start of the integration, the eigenvalues of the diagonally scaled iteration matrix are all contained in one small cluster centered at $(1,0)$. As the stepsize increases, the cluster usually expands. If any circle crosses the axis into the left half-plane, we consider the Gerschgorin ratio to be infinite. Here, we expect that the diagonally scaled iterative method will have difficulty and therefore we do not switch preconditioning off.

Van der Pol's equation (see, for example, [16]) is often used as an example of an ODE system in which transients recur throughout the integration. This second-order equation is frequently rewritten as a system of two first-order ODEs. We have extended it to a large system by using the two ODEs as the reaction terms in a reaction-diffusion PDE system, as follows.

$$(7) \qquad \frac{\partial c^i}{\partial t} = d_i \Delta c^i + f^i(c^1, c^2), \qquad i = 1, 2,$$

$$d_1 = 0.05, \qquad d_2 = 1.0,$$

$$f^1(c^1, c^2) = c^2, \qquad f^2(c^1, c^2) = \eta(1 - (c^1)^2)c^2 - c^1,$$

on the unit square with $\eta = 100$, homogeneous Neumann boundary conditions, and initial conditions

$$c^1(x, y, 0) = 1 - \cos(\pi x)\cos(2\pi y),$$

$$c^2(x, y, 0) = 1 + 2\cos(2\pi x)\cos(\pi y).$$

The spatial derivatives were discretized using a centered second-order finite difference approximation on an evenly spaced mesh with m subdivisions in each direction, yielding a system of $2(m+1)^2$ ODEs.

We solved this problem with $m = 10$ subdivisions, $t \in [0, 1000]$, relative and absolute error tolerances of 10^{-6} and 10^{-4}, respectively, and various switching strategies. As noted earlier, to be conservative about switching preconditioning off, it seems most appropriate to observe the number of linear iterations over a larger number of Newton iterations than is used in the test to switch it on. We require that the linear solver has taken only one iteration per solve over the last 16 Newton iterations before we consider switching preconditioning off. If we use this criterion only, and no information about the iteration matrices, the code switches preconditioning on 14 times and off 13 times in the interval $[0, 1000]$. We note that the original coupled pair of ODEs has period roughly 163, hence goes through six cycles in this time interval. Each cycle involves two intervals in which the solution is changing rapidly.

To use information about the matrix, a bound on the Gerschgorin ratio is chosen. If the smallest Gerschgorin ratio is less than the bound, then we switch preconditioning off if the requirement on number of iterations per solve is also met. We solved Van der Pol's equation again with the Gerschgorin bound set to two and observed the same number of switches as above. Comparing results from the two tests shows that the switches occur at about the same times in the integration; a reassuring result since the switches should be triggered by the stiffness of the continuous system rather than being artifacts of the code. However, we observe that, without matrix information, the code switches preconditioning off as much as one hundred timesteps earlier than when matrix information is used. The earlier switch seems more effective in the sense that the diagonally scaled method experiences no difficulties after the switch, converging in one or two iterations. This result suggests that using a larger Gerschgorin bound might be appropriate, a conclusion supported by tests using values of four and eight that also yield satisfactory behaviour of the switching strategy. With the value eight, we observe the same results as when no matrix information is used; that is, the condition on number of iterations controls the switch. There is no significant difference in total solution time in any case, including a comparison to doing the whole integration with preconditioning (ILU(1) preconditioning was used in all tests). This is due to the small size of the ODE system. We note that the Gerschgorin ratio in the right half-plane grew as large as 10^3 and that some circles extended into the left half-plane on several Jacobian evaluations.

With $m = 20$ subdivisions (882 equations), $t \in [0, 500]$, and the same error tolerances, the integration time using ILU(1) preconditioning on all iterations was 742 seconds, with 452 seconds spent in linear algebra. The time when switching was used with the Gerschgorin bound set to four was 722 seconds, including 435 seconds of linear algebra. The code switched preconditioning on seven times and off six times. Similar results were obtained when the Gerschgorin bound was set to two.

As the mesh is refined, the diffusion term in the PDE causes the ODE system to become stiffer. The results shown in Table 6 are for a grid with $m = 40$ subdivisions (3362 equations), $t \in [0, 500]$, and the same error tolerances. The code switched ILU(1) preconditioning on if the diagonally scaled method was taking four iterations per solve over the last four Newton iterations, and off if the ILU(1)-preconditioned method took only one iteration per solve over the last 16 Newton iterations. The Gerschgorin bound (G.B.) used in each test is given at the top of the column. Here, the most efficient solution technique is to use ILU(1) preconditioning on all steps. With the Gerschgorin bound set to two, preconditioning was switched on four times and off three times; with bound four, the code switched preconditioning on seven times and off six times. The grid size is sufficiently small that the diffusion terms keep the ODE stiff and preconditioning on all steps is the most efficient technique.

TABLE 6
Van der Pol equation; switching on and off.

	No switches	G.B. 2	G.B. 4
NFCN	7447	7967	8129
NJAC	341	376	375
NNWT	3956	4118	4290
NLIN	9496	10,474	11,214
TIME	4365	4569	4597

For the Van der Pol equation, we expect to encounter recurring transients during the integration, causing the ODE solver to use a small timestep to meet the accuracy requirement. Due to this small timestep, all the eigenvalues of the unscaled Newton iteration matrix will be of moderate size and will occur in one single cluster. Checking the Gerschgorin circles of the unscaled matrix confirms that, in the tests above, whenever the code switched preconditioning off, the circles of the unscaled matrix made up one small cluster. This is not the case for the following test problem, used in [5] and [7]. This PDE system, also of reaction-diffusion type, models ozone production in the stratosphere and has the following form.

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + \frac{\partial}{\partial z}\left(K_v(z)\frac{\partial c^i}{\partial z}\right) - V_h \frac{\partial c^i}{\partial x} + R^i(c^1, c^2, t), \qquad i = 1, 2, \tag{8}$$

$$K_h = 4 \cdot 10^{-6}, \quad K_v(z) = 10^{-8} e^{z/5}, \quad V_h = 0.01,$$

$$R^1(c^1, c^2, t) = -k_1 c^1 - k_2 c^1 c^2 + k_3(t) \cdot 7.4 \cdot 10^{16} + k_4(t) c^2,$$

$$R^2(c^1, c^2, t) = k_1 c^1 - k_2 c^1 c^2 - k_4(t) c^2,$$

$$k_1 = 6.031, \qquad k_2 = 4.66 \cdot 10^{-16},$$

$$k_3(t) = \begin{cases} \exp[-22.62/\sin(\pi t/43200)] & \text{for } t < 43200, \\ 0 & \text{otherwise,} \end{cases}$$

$$k_4(t) = \begin{cases} \exp[-7.601/\sin(\pi t/43200)] & \text{for } t < 43200, \\ 0 & \text{otherwise.} \end{cases}$$

The problem is posed for $x \in [0, 20]$, $z \in [30, 50]$ and $t \in [0, 86400]$ with homogeneous Neumann boundary conditions and initial conditions

$$c^1(x, z, 0) = 10^6 \alpha(x)\beta(z), \qquad c^2(x, z, 0) = 10^{12} \alpha(x)\beta(z),$$

$$\alpha(x) = 1 - (0.1x - 1)^2 + (0.1x - 1)^4/2,$$

$$\beta(z) = 1 - (0.1z - 4)^2 + (0.1z - 4)^4/2.$$

If the differential equations are discretized using central differences for both the diffusion and convection terms, as is done in [5] and [7], the eigenvalues of the Jacobian are found in two clusters, as discussed in [5].

The problem was solved with $m = 9$ subdivisions (200 equations) and relative and absolute error tolerances of 10^{-5} and 10^{-3}, respectively. With the Gerschgorin bound set to two, the code switches preconditioning on at $t = 13{,}340$, then off at $t = 42{,}064$, and completes the integration successfully using the diagonally scaled method. Checking the Gerschgorin circles of the unscaled matrix reveals that when preconditioning is switched off, the circles make up two small clusters although the ratio of the extreme limits of all the circles is over 100. In this case, the diagonal scaling has mapped these two clusters into a single small circle. With the bound set to four, the code switches preconditioning on, then off, twice between $t = 13{,}340$ and $t = 35{,}002$, finishing the integration with the diagonally scaled method. It is less efficient than when the bound is set to two.

The results in Table 7 are obtained with a discretization having $m = 19$ subdivisions (800 equations). The ODE solver has some difficulty with the integration, probably due to the central difference discretization, since it does not occur when upstream differencing is used for the convection term. With the Gerschgorin bound set to either two or four, the code switches preconditioning on five times and off five times between $t = 7734$ and t around 42,700; the switches occur at slightly different times depending on the bound. When preconditioning is switched off at $t = 42{,}700$, the Gerschgorin circles of the unscaled Newton iteration matrix form two small clusters. The earlier switches occur because the timestep is small enough, often after a failed step, to force all the circles of the unscaled matrix into one cluster. It is not clear if these switches should be made; the timestep tends to increase rapidly after a failed step and the possible higher accuracy of the more powerful preconditioner might be useful.

TABLE 7
Ozone production model; 800 *equations.*

	No switches	G.B. 2	G.B. 4
NFCN	6001	6267	6179
NJAC	297	320	310
NNWT	3552	3606	3612
NLIN	3552	4985	4865
TIME	654	657	650

The results in Table 8 are obtained with $m = 39$ subdivisions (3200 equations). When the Gerschgorin bound is set to two, the code switches preconditioning on at $t = 7216$, then off at $t = 42{,}321$. With bound four, there are four on-off switches between $t = 7216$ and $t = 42{,}293$, and the integration again finishes using the diagonally scaled method. In this case, it is more efficient to switch preconditioning off when possible.

Overall, it seems that our strategy for switching preconditioning off does detect problem information correctly, i.e., nonstiffness of the Van der Pol oscillator. Also, the diagonal scaling is able to exploit the clustering of the eigenvalues in (8). Computing the six different Gerschgorin ratios can have an effect on when the switches occur. For the Van der Pol equation, all six values tend to be close on all Jacobian evaluations. For (8), where the components are of widely differing magnitudes, the six values can be very different particularly when they are large. When a Gerschgorin bound of two was used to switch preconditioning off, we found that the ratios were close when a switch occurred.

TABLE 8
Ozone production model; 3200 *equations.*

	No switches	G.B. 2	G.B. 4
NFCN	7713	7686	7766
NJAC	307	318	314
NNWT	5182	5065	5173
NLIN	5182	7490	7874
TIME	4559	4113	4168

In contrast, with a bound of four, some switches would not have occurred if all six values had not been computed. Any strategy for making a switch should certainly be conservative. A bound of two seems a better choice, although the value four yields better results in some cases.

4. Adaptive preconditioning in type-insensitive codes. In this section, we insert our strategies for adaptive preconditioning into the type-insensitive ODE solver described in §2. As noted, our code is based on the ideas of Petzold [16]. The method starts integrating with the Adams formulas and monitors the timestep, the eigenvalues of the Jacobian (indirectly), and the region of absolute stability. It switches to the BDFs when it estimates that it can increase the stepsize by a factor of five or more by doing so. When integrating with the BDFs, the code continues to monitor the same information (using the norm of the Jacobian matrix which is now available directly) and will switch back to the Adams formulas if it expects to maintain the same stepsize after the switch.

Our "combined" code incorporates our adaptive preconditioning strategy into the type-insensitive method. The code starts integrating with the Adams formulas and functional iteration and switches to using the BDFs and Newton's method as described above. When Newton's method is first selected, the code uses the iterative linear solver with diagonal scaling only. The performance of the linear solver is monitored and the code may select a more expensive preconditioner as described in §3. If the ODE is only mildly stiff, expensive preconditioning might never be needed.

The strategies for switching preconditioning off and switching from the BDFs to the Adams formulas are independent. The Adams formulas can be selected at any time when the BDFs are in use, regardless of the linear solver currently being used by the code. That is, if a transient component recurred abruptly in the course of the integration, the code could switch directly from the BDFs with the expensive preconditioner to the Adams formulas and functional iteration. In the numerical tests in §4.3, we will see that, for the van der Pol problem, the code actually switches preconditioning off before switching from BDFs to Adams formulas, indicating that it detects a region of "mild" stiffness in the transition from the stiff regime to the transient regime.

We report the results of numerical experiments with our combined code and several test problems. In these tests, the maximum number of linear iterations is always set to 10. Preconditioning is switched on if the diagonally scaled method took four iterations per solve over the last four Newton iterations and off if the Gerschgorin bound (2, unless otherwise specified) is met and only one iteration per solve was required over the last 16 Newton iterations.

In the tables, the preconditioning strategy (either ILU(0) or ILU(1)) and number of equations are shown in the caption. The column labels have the following meanings.

- No switch: BDF method; ILU preconditioning on all iterations.
- Switch: BDF method; switching from diagonal scaling to ILU preconditioning.
- Type-ins: Type-insensitive method; ILU preconditioning on all iterations.
- Combined: Combined method; type-insensitive and switching from diagonal scaling to ILU preconditioning.

4.1. Reaction-diffusion equation in three space dimensions. Equation (5) was solved on a number of different grids ($m = 9$, 2000 equations; $m = 14$, 5488 equations; $m = 20$, 16000 equations) with $\alpha = 100$ and $t \in [0, 100]$. All other problem and method parameters are the same as in §3. The results are shown in Tables 9–14. When $\alpha = 100$, the eigenvalues of the Jacobian are somewhat more widely spread than in the tests reported in §3. Extending the range of integration from $t = 10$ to $t = 100$ simply causes the code to take one large last step, of order 100 itself. This large step poses no problem for the iterative solver in this case.

TABLE 9
ILU(0) *preconditioning;* 2000 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1414	1335	1448	1444
NJAC	65	60	44	44
NNWT	613	596	350	346
NLIN	653	726	391	449
TIME	298	262	237	226

TABLE 10
ILU(1) *preconditioning;* 2000 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1414	1334	1439	1444
NJAC	65	60	42	44
NNWT	613	595	365	346
NLIN	631	697	379	437
TIME	328	259	260	227

TABLE 11
ILU(0) *preconditioning;* 5488 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1365	1378	1278	1270
NJAC	62	64	44	44
NNWT	602	589	400	392
NLIN	674	722	465	530
TIME	869	798	725	670

TABLE 12
ILU(1) *preconditioning; 5488 equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1365	1378	1290	1270
NJAC	62	64	44	44
NNWT	602	589	412	392
NLIN	643	698	445	501
TIME	966	801	763	667

TABLE 13
ILU(0) *preconditioning; 16000 equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1401	1363	1568	1527
NJAC	64	60	43	42
NNWT	612	624	372	343
NLIN	744	849	511	556
TIME	2895	2698	2541	2365

TABLE 14
ILU(1) *preconditioning; 16000 equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1413	1362	1568	1539
NJAC	65	60	43	43
NNWT	612	623	372	343
NLIN	694	786	443	505
TIME	3158	2668	2662	2342

We always find that the method that combines the type-insensitive approach with adaptive preconditioning is the most efficient, saving 18 to 24% of execution time with ILU(0) preconditioning, and 26 to 31% with ILU(1) preconditioning. Without adaptive preconditioning, the ILU(0) method is more efficient, but when preconditioning is applied only when necessary, it is just as efficient to use the more powerful ILU(1) approach.

4.2. Ozone production model. In §3, (8) was discretized using centered differences for both the diffusion and convection terms, following [5] and [7]. Since the convection coefficient $V_h = 0.01$ while the diffusion coefficients have magnitude 10^{-6} to 10^{-4}, this problem is convection-dominated and it is reasonable to use upstream differencing for the convection term. This discretization leads to a Jacobian matrix that is better conditioned than when centered differences are used and allows the code to take much larger timesteps. We do not expect that preconditioning will be switched off once it has been turned on. Upstream differencing was used in computing the results in Tables 15 and 16. The equations were discretized on a grid with 41 mesh points in each direction giving

$\Delta x = \Delta z = 0.5$ and a system of 3362 equations. As before, an absolute error tolerance of 10^{-3} and a relative tolerance of 10^{-5} were used.

TABLE 15
ILU(0) *preconditioning; upstream differences;* 3362 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1508	1566	1533	1375
NJAC	78	81	70	62
NNWT	705	731	599	523
NLIN	954	1083	826	773
TIME	677	696	652	574

TABLE 16
ILU(1) *preconditioning; upstream differences;* 3362 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1276	1614	1359	1350
NJAC	61	86	57	59
NNWT	647	729	557	528
NLIN	837	1004	800	775
TIME	613	709	618	590

Again, it is found that the method that combines the type-insensitive approach with adaptive preconditioning is the most efficient, although the saving is not as significant as for (5). The method with ILU(1) preconditioning on all steps does well, taking fewer function evaluations than any other approach. As mentioned previously, this seems to indicate that there is an advantage in accuracy to using ILU(1) preconditioning. Adaptive preconditioning alone degrades efficiency although it does well in combination with the type-insensitive method. With the combined method, the switch from the Adams formulas to the BDFs occurs at step 93, $t = 3.93$, then preconditioning is switched on at step 140, $t = 1142$. With adaptive preconditioning only, the switch occurs later at step 189, $t = 2167$.

In Table 17, we show results computed with the type-insensitive and combined methods and a central difference discretization of all terms, for comparison with results given in §3. (Those results are also included in this table.) In this case, the switching method and the combined method have comparable cost. In the switching method, preconditioning is turned on at $t = 7216$, then off at $t = 42,321$. In the combined method, the BDFs are selected as above at $t = 3.93$, step 93; preconditioning is turned on at $t = 5837$, and off at $t = 42{,}360$.

4.3. Van der Pol equation. We use the modified Van der Pol equation (7) to investigate the interaction between turning preconditioning on and off and switching from the BDFs back to the Adams formulas. As we noted in §3, on a grid with $m = 40$ subdivisions (3362 equations), the ODE system is quite stiff due to the effect of the diffusion terms regardless of the behaviour of the reaction terms. The results shown in Table 18 were computed using the same problem and method parameters as in §3 on the interval

TABLE 17
ILU(1) *preconditioning; centered differences;* 3200 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	7713	7686	7933	7668
NJAC	307	318	311	296
NNWT	5182	5065	5157	5016
NLIN	5182	7490	5157	7347
TIME	4559	4113	4863	4146

$[0, 500]$. Recall that the underlying small ODE system goes through three cycles in this interval. G.B. denotes the Gerschgorin bound.

TABLE 18
ILU(1) *preconditioning;* 3362 *equations.*

	No switch	Switch		Type-ins	Combined	
		G.B. 2	G.B. 4		G.B. 2	G.B. 4
NFCN	7447	7967	8129	7840	7521	8054
NJAC	341	376	375	353	334	359
NNWT	3956	4118	4290	4015	3890	4169
NLIN	9496	10,474	11,214	9608	9073	11,198
TIME	4365	4569	4597	4485	4267	4639

In the "switching" code, preconditioning is first turned on at time $t = 0.025$, step 65. Since the total integration takes around 4000 steps, this is very early in the integration. With the Gerschgorin bound set to two, preconditioning is switched on four times and off three times; with bound four, the code switches preconditioning on seven times and off six times during the integration. The type-insensitive method is less effective than the code that uses ILU(1) preconditioning throughout, somewhat surprisingly, since the type-insensitive code switches from the Adams formulas to the BDFs at time $t = 0.0076$, step 85, and never switches back to the Adams formulas. The combined method with Gerschgorin bound 2 turns preconditioning on at $t = 0.026$, step 115, and never switches it off. With the bound set to 4, the combined method makes the same number of switches as the "switching" code, at essentially the same times in the integration.

This problem is best solved with ILU(1) preconditioning. The same set of tests was run using ILU(0) preconditioning and the same relative performances were observed. The total time to complete the integration was about 1000 seconds greater in all cases.

To investigate the effect of switching between the Adams formulas and the BDFs when the number of equations is large, the diffusion coefficients were reduced from 0.05 and 1 to 0.005 and 0.1, respectively. The numerical results using ILU(0) preconditioning are given in Table 19, those using ILU(1) preconditioning in Table 20. Other parameters of the problem and method are the same as in §3, with $t \in [0, 500]$.

With ILU(0) preconditioning, the "switching" method and the combined method turn preconditioning on seven times and off six times for both choices of Gerschgorin bound. The type-insensitive method selects the BDFs for the first time at $t = .709$, step

TABLE 19
ILU(0) *preconditioning;* 3362 *equations; small diffusion.*

	No switch	Switch		Type-ins	Combined	
		G.B. 2	G.B. 4		G.B. 2	G.B. 4
NFCN	9503	9363	9579	9918	9930	10,129
NJAC	414	398	408	275	262	275
NNWT	5264	5288	5402	3171	2941	2990
NLIN	7964	8517	8792	5889	5849	6129
TIME	4216	4048	4073	3492	3343	3391

TABLE 20
ILU(1) *preconditioning;* 3362 *equations; small diffusion.*

	No switch	Switch		Type-ins	Combined	
		G.B. 2	G.B. 4		G.B. 2	G.B. 4
NFCN	9519	9605	9612	10,049	10,041	9966
NJAC	417	411	413	288	275	266
NNWT	5250	5398	5385	3302	2954	2939
NLIN	6547	7472	7761	4703	4570	4934
TIME	4028	3879	3853	3360	3132	3096

1285, where the total integration still takes around 4000 steps. Following the initial formula change, there are three subsequent switches to the Adams formulas then back to the BDFs. The combined method makes four subsequent switches between Adams formulas and BDFs. Three of these switches are at essentially the same values of t as the type-insensitive method, but there is one additional switch. There are two switches between preconditioning and diagonal scaling where the Adams formulas are not selected. A switch to the Adams formulas is always preceded by switching preconditioning off, typically by about 80 to 100 timesteps.

When ILU(1) preconditioning is used, the "switching" method and combined method still turn preconditioning on seven times and off six times for both choices of Gerschgorin bound. The type-insensitive method of course selects the BDFs for the first time at the same timestep as in the ILU(0) case, then there are four subsequent switches to the Adams formulas then back to the BDFs. The combined method makes the same switches, with either Gerschgorin bound. A switch to the Adams formulas is always preceded by switching preconditioning off; the switches occur at essentially the same times for ILU(0) preconditioning.

The strategy for formula selection appears to be robust with respect to effects of the linear algebra, as one would hope. Also, the switches between diagonal scaling and preconditioning always occur at about the same t values, indicating that the switching strategy is, in fact, detecting properties of the ODE system. It is, though, difficult to choose a value for the Gerschgorin bound based on these tests. As noted in §3, the value 2 appears preferable since it gives a saving whenever one is achieved by switching preconditioning and does better when a saving is not achieved.

4.4. A "dense" two-dimensional problem. The system of PDEs defined by (9) below is used as a test problem in [6]. Although this problem is similar in nature to (5)—both are reaction-diffusion systems modelling a predator-prey interaction—it poses a slightly different challenge to the iterative methods since it includes a large number of species, making the associated Jacobian matrix relatively dense.

$$\frac{\partial c^i}{\partial t} = d_i \Delta c^i + f^i(c), \qquad i = 1, 2, \ldots, s, \tag{9}$$

where $c = (c^1,\ c^2, \ldots, c^s)^T, p = s/2,$

$$\begin{aligned}
d_i &= 1, \qquad i = 1, 2, \ldots, p,\\
d_i &= 0.05, \qquad i = p+1, \ldots, s,\\
f^i(c) &= c^i \left(b_i + \sum_{j=1}^{s} a_{ij} c^j \right),\\
a_{ii} &= -1 \quad \forall i,\\
a_{ij} &= -0.5 \cdot 10^{-6}, \quad i \le p, \quad j > p,\\
a_{ij} &= 10^4, \quad i > p, \quad j \le p,\\
a_{ij} &= 0 \quad \text{for all other } i \text{ and } j;\\
b_i &= (1 + \alpha xyz), \quad i \le p,\\
b_i &= -(1 + \alpha xyz), \quad i > p.
\end{aligned}$$

The system (9) is defined on the unit square with $t \in [0, 10]$. The problem has homogeneous Neumann boundary conditions and initial conditions

$$c^i(x, y, 0) = 10 + i[16x(1-x)y(1-y)]^2, \qquad 1 \le i \le s.$$

Here, we take $s = 20$ ($p = 10$) and $\alpha = 50$. The steady state solution is spatially inhomogeneous and the Jacobian is highly nonsymmetric at equilibrium [6].

The spatial derivatives are discretized using a centered second-order finite difference approximation on an evenly spaced mesh with m subdivisions in each direction, yielding a system of $s(m+1)^2$ ODEs. Following [6], we take $m = 11$, which gives an ODE system of 2880 equations. This problem is solved using a relative error tolerance of 10^{-6} and an absolute error tolerance of 10^{-8}. Numerical results for ILU(0) and ILU(1) preconditioning are given in Tables 21 and 22, respectively.

The approaches that use ILU(0) preconditioning are always more efficient than those using ILU(1) preconditioning, due to the density of the Jacobian matrix, which has 42,240 nonzero entries, as does the ILU(0) preconditioner, while the ILU(1) preconditioner has 112,840 nonzeros. Hence, even though the code using ILU(1) preconditioning generally uses fewer function and Jacobian evaluations, nonlinear and linear iterations, the total CPU time is greater. In both cases, the combined method is the most efficient.

Brown and Hindmarsh [6] identify the nonstiff transient region for this problem as roughly $0 \le t \le 10^{-3}$. Our strategy for switching on preconditioning also picks out this region. While the switch from the Adams formulas to the BDFs takes place at $t = 2.88 \cdot 10^{-5}$, the switch from diagonal scaling to preconditioning takes place at

TABLE 21
ILU(0) *preconditioning;* 2880 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1614	1515	1388	1330
NJAC	43	40	31	29
NNWT	395	380	203	201
NLIN	780	755	655	621
TIME	507	449	416	384

TABLE 22
ILU(1) *preconditioning;* 2880 *equations.*

	No switch	Switch	Type-ins	Combined
NFCN	1451	1592	1257	1173
NJAC	38	42	27	24
NNWT	372	401	184	186
NLIN	586	638	397	496
TIME	570	507	422	418

$t = 2.45 \cdot 10^{-3}$ when the "switching" code is used, and at $t = 2.76 \cdot 10^{-3}$ in the combined code.

The ratio of the number of linear iterations to the number of Newton iterations (AVDIM) is reported in [6]. This value provides a rough comparison of the work being done by the iterative method in the various approaches. For problem (9), it is also interesting to compare the work done in the nonstiff and stiff regimes, which we identify with the intervals $[0, 1]$ and $[1, 10]$, respectively, for convenience. Values are reported in Table 23. As expected, AVDIM increases when diagonal scaling is used and also when the type-insensitive method is used. The iterative solver works considerably harder in the stiff regime. The values for AVDIM found here are similar to those reported in [6].

TABLE 23
AVDIM: ratio of linear to Newton iterations.

ILU(0)		No switch	Switch	Type-ins	Combined
	$[0, 1]$	1.82	1.90	2.88	2.97
	$[1, 10]$	5.17	5.10	5.72	4.53
ILU(1)					
	$[0, 1]$	1.48	1.53	1.99	2.56
	$[1, 10]$	5.10	4.22	4.58	4.25

5. Conclusions. We have developed a strategy for the adaptive choice of preconditioning when an iterative linear solver is used in an ODE code. This strategy successfully identifies characteristics of the problem and is able to switch preconditioning on or off according to those characteristics. This approach combines naturally with the use of a type-insensitive ODE solver. For the test problems considered, we found that, when the type-insensitive approach yields a saving in execution time, the combined method increases the saving. There are problems for which the adaptive approach is not effective; in these cases, it appears that the type-insensitive approach is not appropriate either. As a simple rule of thumb, the combined method is effective when both the Adams formulas and the diagonally scaled preconditioning are used over a significant percentage of steps.

There is a variety of ways of adapting the preconditioner that one might consider. Possible choices for the "cheap" preconditioner were discussed in the introduction. Another idea is to adapt through various more powerful preconditioners, say ILU(0) to ILU(1), possibly to a direct solution method if the ODE appears very difficult to solve. However, the difference in cost of the various preconditioning methods is not that great for a wide class of problems. As we noted earlier, for example, ILU(0) preconditioning is only about 40% more expensive than diagonal scaling on many problems derived from PDEs. This suggests that little additional saving would be achieved by this incremental approach. A "cheap" preconditioner that was effective over a longer range would be more useful. Switching to a direct method is possible if the problem size is not too great or if a sparse direct method does not suffer too much fill-in.

Our strategy exploits the availability of the Jacobian matrix during intervals of the integration in which the expensive preconditioner is in use, both in the choice of preconditioner and in the strategy for switching preconditioning off. Our choice of preconditioner was motivated by the fact that ILU factorization can be applied without knowledge of a particular matrix's properties, i.e., it can be supplied to a user as part of a "black-box" code. Also, implementations of these preconditioners were readily available to us. Our idea of switching between preconditioners that have different costs is independent of the choice of preconditioner, although parameters in the strategy would have to be adjusted to suit the relative costs. The problem of switching preconditioning off if Jacobian information is not available is somewhat more difficult. One possibility would be to use information about numbers of iterations only. The code might switch preconditioning off if a *reduction* in the number of iterations used by the preconditioned iterative method was noted and the number of linear iterations remained small over many Newton iterations. In addition, the code could have a flag that disabled switching off entirely. If the strategy was fooled, i.e., preconditioning had to be turned on again immediately after being turned off, the flag would be set. Another possibility is to use the Gerschgorin circles of the preconditioner in the switching strategy, since the preconditioner is an approximation to the Newton iteration matrix, although admittedly a rough one. In many cases, computing a finite-difference approximation to a sparse Jacobian matrix is, in fact, a relatively cheap operation and hence basing a preconditioning strategy on explicit Jacobian information is not unreasonable.

Both the ILU(0) and ILU(1) preconditioners worked well for these test problems. The choice between them depends on the amount of fill-in generated by the ILU(1) factorization and how difficult the ODE is to solve. As we saw for the last test problem, even though ILU(1) preconditioning can be significantly more efficient in terms of the number of linear iterations, the cost per iteration may make ILU(0) preferable. We note that the choice of the maximum number of iterations can have a significant effect on the overall performance of the code. In a method such as Orthomin or GMRES, the amount of storage restricts the choice of this maximum. (Although restarting can be used, it is not clear how effective it is.) If the ODE solver fails and reduces the stepsize because the iterative method did not converge, this can have a ripple effect throughout the rest of the

integration, significantly increasing the total number of steps and amount of work. The maximum should be chosen as large as possible, consistent with the storage available.

REFERENCES

[1] A. BEHIE AND P. FORSYTH, *Comparison of fast iterative methods for symmetric systems*, IMA J. Numer. Anal., 3 (1983), pp. 41–63.

[2] ———, *Incomplete factorization methods for fully implicit simulation of enhanced oil recovery*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 543–561.

[3] M. BERZINS AND R. FURZELAND, *A user's manual for SPRINT—a versatile software package for solving systems of algebraic, ordinary and partial differential equations: Part 1—Algebraic and ordinary differential equations*, Tech. Rep. TNER.85.058, Thornton Research Centre, Shell Research Ltd., Thornton, UK, 1985.

[4] ———, *An adaptive method for the solution of stiff and non-stiff differential equations*, Tech. Rep., School of Computer Studies, Leeds Univ., Leeds, UK, 1990.

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

[6] ———, *Reduced storage matrix methods in stiff* ODE *systems*, Appl. Math. Comp., 31 (1989), pp. 40–91.

[7] G. BYRNE, *Pragmatic experiments with Krylov methods in the stiff* ODE *setting*, in Proc. IMA Conf. Computational Ordinary Differential Equations, 1989, J. Cash and I. Gladwell, eds., Oxford University Press, London, 1992.

[8] T. CHAN AND K. JACKSON, *The use of iterative linear-equation solvers in codes for large systems of stiff* IVP*s for* ODE*s*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 378–417.

[9] E. D'AZEVEDO, P. FORSYTH, AND W. TANG, *Towards a cost-effective* ILU *preconditioner with high level fill*, BIT, 32 (1992), pp. 442–463.

[10] R. DEMBO, S. EISENSTAT, AND T. STEIHAUG, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[11] C. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[12] C. GEAR AND Y. SAAD, *Iterative solution of linear equations in* ODE *codes*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 583–601.

[13] I. GUSTAFSSON, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[14] A. HINDMARSH, LSODE *and* LSODI, *two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, 1980, pp. 10–11.

[15] S. NORSETT AND P. THOMSEN, *Switching between modified Newton and fix-point iteration for implicit* ODE-*solvers*, BIT, 26 (1986), pp. 339–348.

[16] L. PETZOLD, *Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 136–148.

[17] L. SHAMPINE, *Type-insensitive* ODE *codes based on implicit A-stable formulas*, Math. Comp., 36 (1981), pp. 499–510.

[18] ———, *Type-insensitive* ODE *codes based on implicit $A(\alpha)$-stable formulas*, Math. Comp., 39 (1982), pp. 109–123.

[19] A. SHERMAN AND A. HINDMARSH, *GEARS, a package for the solution of sparse, stiff ordinary differential equations*, in Electrical Power Problems: the Mathematical Challenge, A. Erisman, K. Neves, and M. Dwarakanath, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1981, pp. 190–200.

[20] P. SONNEVELD, CGS, *a fast Lanczos-type solver for nonsymmetric systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[21] H. VAN DER VORST AND P. SONNEVELD, CGSTAB, *a more smoothly converging variant of* CGS, Tech. Rep. 90–50, Delft Univ. of Technology, Delft, the Netherlands, 1990.

[22] P. VINSOME, *Orthomin, an iterative method for solving sparse sets of simultaneous linear equations*, in Society of Petroleum Engineers of AIME, Paper SPE 5729, 1976.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 824–859, July 1993 005

NUMERICAL METHODS FOR HYPERBOLIC CONSERVATION LAWS WITH STIFF RELAXATION II. HIGHER-ORDER GODUNOV METHODS*

RICHARD B. PEMBER†

Abstract. A higher-order Godunov method is presented for hyperbolic systems of conservation laws with stiff, relaxing source terms. The goal is to develop a Godunov method that produces higher-order accurate solutions using time and space increments governed solely by the nonstiff part of the system, i.e., without fully resolving the effect of the stiff source terms. It is assumed that the system satisfies a certain "subcharacteristic" condition. The method is a semi-implicit form of a method developed by Colella for hyperbolic conservation laws with nonstiff source terms. In addition to being semi-implicit, our method differs from the method for nonstiff systems in its treatment of the characteristic form of the equations. The method is applied to a model system of equations and to a system of equations for gas flow with heat transfer. Our analytical and numerical results show that the modifications to the nonstiff method are necessary for obtaining second-order accuracy as the relaxation time tends to zero. Our numerical results also suggest that certain modifications to the Riemann solver used by the Godunov method would help reduce numerical oscillations produced by the scheme near discontinuities. The development of a modified Riemann solver is a topic of future work.

Key words. hyperbolic conservation laws with relaxation, stiff source terms, finite difference methods, Godunov-type methods

AMS subject classifications. 35L65, 65M25, 76.65

1. Introduction. Hyperbolic conservation laws with relaxation appear in the study of a variety of physical phenomena, for example, in the modeling of thermally and chemically nonequilibrium fluid flows. The term "hyperbolic system of conservation laws with relaxation" is used here in the sense of Whitham [42], [43] and Liu [27] to denote a hyperbolic system of N partial differential equations in conservation form with source terms which has as a limit a hyperbolic system of M equations, $M < N$, called the equilibrium system as $N - M$ relaxation time parameters $\tau_i \rightarrow 0$. We call the system of N equations the nonequilibrium system. The characteristic speeds of the nonequilibrium system are called frozen characteristic speeds, while those of the equilibrium system are called equilibrium characteristic speeds. We say that a system of conservation laws with relaxation is stiff when at least one of its relaxation times is small compared to the time scale determined by the frozen characteristic speeds of the system and some appropriate length scale. In this paper we present a higher-order Godunov method for hyperbolic systems of conservation laws with stiff, relaxing source terms. Our goal is to develop a second-order Godunov method which produces higher-order accurate solutions using time and space increments governed solely by the nonstiff part of the system, i.e., without fully resolving the effect of the stiff source terms. We base our development on the higher-order Godunov method for hyperbolic conservation laws with nonstiff source terms presented by Colella [11].

We assume that the system of conservation laws with relaxation contains a single rate equation, i.e., $M = N - 1$. We also assume that the frozen and the equilibrium characteristic speeds of the system alternate in a particular way [27], [43]. Specifically, suppose that the dependent variables of the nonequilibrium system are $u_1, \ldots, u_N$, and

*Received by the editors February 12, 1992; accepted for publication (in revised form) July 27, 1992.

†Computing and Mathematics Research Division, Lawrence Livermore National Laboratory, L-316, P.O. Box 808, Livermore, California 94550 (pember@llnl.gov). This work was performed under the auspices of the U.S. Department of Energy, by the Lawrence Livermore National Laboratory, under contract W-7405-Eng-48. Partial support was provided by the Applied Mathematical Sciences Program of the Office of Energy Research.

that u_N is the dependent variable governed by a rate equation, i.e. the Nth equation of the system has a source term which acts to restore u_N to equilibrium. The equilibrium system is then found by substituting the equilibrium value of u_N, $u_*(u_1, \ldots, u_M)$, in the first M equations of the nonequilibrium system. We then say that the frozen characteristic speeds λ_i and the equilibrium characteristic speeds μ_i satisfy the *subcharacteristic condition* if they satisfy the inequality

$$\lambda_1 < \mu_1 < \lambda_2 < \mu_2 \cdots < \mu_M < \lambda_N, \tag{1.1}$$

where the λ_i are evaluated at the equilibrium state $(u_1, \ldots, u_M, u_*(u_1, \ldots, u_M))$. Hyperbolic systems of equations with relaxation whose frozen and equilibrium characteristic speeds satisfy (1.1) include the equations describing gas flow with vibrational nonequilibrium and with nonequilibrium due to chemical dissociation [39].

There are a number of theoretical results in the literature based on the subcharacteristic condition. Whitham [42], [43] shows that the condition is necessary for stability in the case of linearized systems with relaxation. Liu [27] shows for $N = 2$ that if the frozen and the equilibrium characteristic speeds always satisfy the subcharacteristic condition, then the corresponding equilibrium equation is stable under small perturbations and the time-asymptotic solutions of the system are completely determined by the equilibrium equation. Chen, Levermore, and Liu [8] show that if the subcharacteristic condition is always satisfied, then solutions of the system tend to solutions of the equilibrium equation as the relaxation time tends to zero.

In this paper we develop a higher-order Godunov method for solving hyperbolic systems of conservation laws with stiff relaxation under the assumption that the frozen and the equilibrium characteristic speeds of the system satisfy (1.1). Our results in an earlier paper [31] show that under this assumption one can always obtain first-order accurate solutions of these systems with a Godunov-type method which does not fully account for the effect of the stiff source terms. Four new issues must be addressed, however, in the development of a higher-order accurate Godunov method for hyperbolic conservation laws with stiff relaxation which incorporates the methodology in [11]. Two of the issues are relevant to other shock-capturing methods. (See the discussion by Yee and Shinn [46].) One issue is whether the method should be fully implicit or semi-implicit, i.e., implicit only with respect to the stiff source term. The second issue is if the method can use Strang splitting [33] to integrate the source term and the conservation laws in separate, fractional steps and still produce higher-order accurate results. A third issue is relevant to all Godunov-type methods. Godunov's original method [18] and subsequent Godunov-type methods (see [38], [44], and [45] for overviews) use either the exact or the approximate solution of Riemann problems to calculate numerical fluxes. The issue, then, is if the "Riemann solver" needs to account for the equilibrium equation and/or the presence of source terms. The fourth issue is specific to the higher-order Godunov method of Colella [11]. In this method the higher-order accuracy results from a step which uses the characteristic form of the equations. The step discards certain components of these equations depending on the signs of the characteristic speeds. The last issue, then, is if one needs to modify this step (and, if so, how) to account for the presence of stiff source terms and for the fact that waves propagate at the equilibrium characteristic speeds as the relaxation time approaches zero.

(*Remark.* The issue of whether to use a fractional step or an unsplit approach is not actually an issue per se in applying the methodology in [11]; the approach there is an unsplit one. We address this issue because there is a lack of consensus in the literature on this issue; see discussions in [16], [23], and [46].)

We resolve the first of these four issues by assuming that the stiffness in the equations is due entirely to the source term. Hence, a fully implicit method is unnecessary for stability [46], and we choose to use a semi-implicit approach for computational simplicity. In the remainder of this paper we address the other issues with analytical and numerical results. We present evidence in support of the following conclusions:

(1) The method must use an unsplit approach which couples the source term and the conservation laws in order to achieve higher-order accuracy for stiff systems.

(2) The Riemann solver does not have to account for the equilibrium equation nor for the presence of source terms for the purpose of accuracy. Nevertheless, the method should use a Riemann solver which satisfies the following:

(1.2)
(A) The Riemann solver reduces to a Riemann solver for the equilibrium equation in the limit as relaxation time tends to zero.
(B) The numerical flux determined by the Riemann solver implies that the method is an upwind scheme [21] for all positive values of the relaxation time.

A Riemann solver satisfying (1.2) ensures that the method reduces to an upwind-centered difference scheme [35] for the equilibrium equation in the limit as the relaxation time approaches zero. If the Riemann solver does not satisfy (1.2), the method is more likely to produce oscillations at discontinuities [35], [40], [41] when the system is stiff.

(3) The step of the higher-order Godunov method which uses the characteristic form of the equations does not have to account directly for the characteristic form of the equilibrium equations. However, the test for determining which components of the characteristic form of the nonequilibrium equations to discard must be modified in order to achieve higher-order accuracy for stiff systems. This change in turn requires a modification to the portion of the algorithm that helps preserve monotonicity [37].

In this paper we present a higher-order Godunov method that uses an unsplit approach and that implements a new test for determining which components of the characteristic form of the nonequilibrium equations to discard. Both of these features are needed to ensure that the method reduces to a second-order method for the equilibrium system as the relaxation time tends to zero.

Our method, however, does not actually use a Riemann solver which satisfies (1.2). We believe that the development of such a Riemann solver for a general hyperbolic conservation law with relaxation is an open problem. Bolstad et al. [4] discuss a semi-implicit, unsplit second-order Godunov method for a stiff system of equations modeling gas dynamics with heat sources and sinks. The Riemann solver in their method uses shock jump conditions which are correct for zero and infinite relaxation times, and which vary in a continuous way for intermediate values of the relaxation time. We are uncertain, however, whether their method is an upwind scheme for all intermediate values of the relaxation time.

The remainder of the paper is organized as follows:

In §2 we review the second-order Godunov method [1], [11] for hyperbolic conservation laws with nonstiff source terms in one space dimension. In §3 we discuss how to modify the method in [11] in order to obtain a second-order Godunov method for hyperbolic conservation laws with stiff, relaxing source terms. We refer to the methods in §§2 and 3 as the *nonstiff method* and the *frozen method*, respectively. In §4, we briefly present three variant methods. We present these methods not as potential alternatives to the frozen method but as aids in the analysis of the frozen method.

In §5 we develop a model system of equations and two specific test problems. In §6, we analyze the frozen method and the three variant methods presented in §4. We show that the frozen method is second-order accurate for smooth solutions in the limit as the relaxation time approaches zero. We also demonstrate that two of its features, the coupling of the source term and its modified treatment of the characteristic form of the equations, are necessary for obtaining higher-order accurate solutions when the model system is stiff. We also show that the method does not reduce to an upwind method for the equilibrium equation as the model system becomes increasingly stiff. In §7 we show numerical results which validate the analysis in §6.

In §8 we apply the frozen method and the three variant methods to the equations describing one-dimensional inviscid gas flow coupled to a constant temperature heat sink. We solve two test problems, an isothermal shock and an isothermal rarefaction. The results are consistent with the results discussed in §§6 and 7 for the model system. The results in §8 also show that the frozen method may produce numerical oscillations near shocks. We conclude that the numerical oscillations produced by the frozen method may be reduced if its Riemann solver were replaced by one which satisfies (1.2). In §9, we discuss some considerations in the development of a Riemann solver with the property in (1.2). We also discuss extending the method to the equations of gas flow with chemical and vibrational nonequilibrium.

Remark. The method introduced in §3 addresses the problem of obtaining higher-order accuracy for stiff hyperbolic conservation laws with relaxation for which the frozen and the equilibrium characteristic speeds satisfy some form of a subcharacteristic condition. The method does not solve the problem of spurious solutions of hyperbolic systems with relaxation for which the subcharacteristic condition is violated [31], or, more generally, of hyperbolic systems with stiff source terms for which the limits of vanishing viscosity and vanishing relaxation time do not commute. In particular, the method does not alleviate the problem of spurious solutions arising in the numerical approximation of the equations of reacting gas dynamics [13] or any of the model systems for reacting flow [26], [29]. In [31], we conjecture that there are inherent problems in using a purely shock capturing finite difference scheme to solve a system of hyperbolic conservation laws with stiff source terms for which the limits of vanishing viscosity and vanishing relaxation do not commute. The problem of spurious solutions for systems of this type can be solved, we believe, only by effectively resolving the time scales of the stiff source term via methodologies such as adaptive mesh refinement [2], front tracking [5], subcell resolution [20], [7], [19], and numerical induction [15].

2. Review: Method for nonstiff source terms.

In this section, we review the second-order Godunov method [1], [11] for hyperbolic systems of conservation laws with nonstiff source terms in one space dimension. We refer to this method as the *nonstiff method*. This method is based on earlier work by van Leer [36], [37] and by Colella and Woodward [14] for gas dynamics.

We consider a hyperbolic system of conservation laws with nonstiff source terms in one space dimension, i.e., a system of the form

$$\begin{gathered}\frac{\partial U}{\partial t}+\frac{\partial F(U)}{\partial x}=S(U,x), \quad t\geq 0, \quad -\infty<x<\infty, \\ U=U(x,t), \quad F(U), \quad S(U,x)\in \boldsymbol{R}^N,\end{gathered} \tag{2.1}$$

where for all U under consideration the Jacobian of F, $A(U)=\partial F/\partial U$, has N real eigenvalues $\lambda_1\leq\lambda_2\cdots\leq\lambda_N$, called the characteristic speeds of the system, corresponding to N linearly independent right eigenvectors $r^k, k = 1, \dots, N$. We assume that the ordering of the characteristic speeds is global, and that all N characteristic fields are either genuinely nonlinear or linearly degenerate. We also assume that the Riemann problem for (2.1) with $S(U,x) \equiv 0$,

$$\begin{aligned} U(x,0) &= U_L, \quad x < 0, \\ &= U_R, \quad x > 0, \end{aligned}$$

has a unique solution for all U_L, U_R under consideration. To aid in the description of the algorithm, we define the matrix of right eigenvectors of $A(U)$ by $R = (r_1 \cdots r_N)$ and the matrix of left eigenvectors by $L = R^{-1} = (l_1 \cdots l_N)^T$, where R and L have been normalized so that $LR = \boldsymbol{I}$.

We now describe the second-order Godunov method for (2.1). We pick a fixed spatial grid with cell width Δx indexed over j. To advance the solution at time t_n, we use a time step Δt which satisfies the Courant–Friedrichs–Lewy (CFL) stability condition,

$$\Delta t = \sigma \frac{\Delta x}{\max_{k,j} |\lambda^n_{k,j}|}, \tag{2.2}$$

where $\lambda^n_{k,j}$ is the kth eigenvalue of $A(U^n_j)$ and σ is a constant called the Courant number, $\sigma < 1$. We assume that the largest eigenvalue of the Jacobian matrix $\partial S(U,x)/\partial U$ is significantly smaller in magnitude than $(\max_{k,j} |\lambda^n_{k,j}|)/\Delta x$, i.e., the system is not stiff with respect to the source term.

The algorithm consists of four general steps:

(1) the calculation of limited central difference approximations of ΔU in each cell j, where $\Delta U/\Delta x$ is an approximation of $\partial U/\partial x$ at the cell center;

(2) the calculation of time-centered left and right states, $U^{n+1/2}_{j+1/2,l}$ and $U^{n+1/2}_{j+1/2,r}$, at each cell edge, i.e., at $t = (n + 1/2)\Delta t$, $x = (j + 1/2)\Delta x$. Since these values are calculated by integrating the linearized characteristic form of (2.1), we refer to this step as the *characteristic tracing* step;

(3) the solution of the Riemann problem at each cell edge with left and right states given by $U^{n+1/2}_{j+1/2,l}, U^{n+1/2}_{j+1/2,r}$ for (2.1) with $S(U,x) \equiv 0$ and the evaluation of that solution along the ray $x/t = 0$ to obtain $U^{n+1/2}_{j+1/2}$;

(4) the calculation of U^{n+1}_j by an application of the divergence theorem. The solution of the Riemann problem in (3) is used to approximate the flux terms $F^{n+1/2}_{j+1/2} = F(U^{n+1/2}_{j+1/2})$ and to approximate the average of $S(U,x)$ over $[(j-1/2)\Delta x, (j+1/2)\Delta x] \times [t_n, t_n + \Delta t]$:

$$\begin{aligned} U^{n+1}_j &= U^n_j + \frac{\Delta t}{\Delta x}\left(F^{n+\frac{1}{2}}_{j-\frac{1}{2}} - F^{n+\frac{1}{2}}_{j+\frac{1}{2}}\right) \\ &\quad + \frac{\Delta t}{2}\left(S\left(U^{n+\frac{1}{2}}_{j-\frac{1}{2}}, \left(j - \frac{1}{2}\right)\Delta x\right) + S\left(U^{n+\frac{1}{2}}_{j+\frac{1}{2}}, \left(j + \frac{1}{2}\right)\Delta x\right)\right). \end{aligned}$$

We refer to this step as the conservative differencing step. A dissipative flux term can be added to the numerical flux when strong shocks are present; see [11] and [14].

Step (4) is self-explanatory. We refer the reader to the literature for the details of step (3). Exact Riemann problem solvers exist for very few hyperbolic systems. Riemann solvers for gas dynamics [9], [12] and for gas dynamics with combustion [10], [34] have been developed. There are also a number of algorithms [1], [30], [32] for approximating the solution of the Riemann problem for general hyperbolic systems. We describe the details of the other two steps below. Throughout the description we often omit the superscript n to simplify the notation.

Remark. Steps (1)–(3) of the algorithm can be applied to any W that is related to U by an invertible map $U = U(W)$. For example, in the case of Eulerian gas dynamics in one space dimension where $U = (\rho, \rho u, \rho E)$, one can use $W = (\rho, u, p)$.

Remark. We want to stress here that there is no reason for the purpose of accuracy to include the effect of the source term in the solution of the Riemann problem. The left and right values calculated in the characteristic tracing step at a given edge are locally second-order accurate estimates of the solution at that edge at the half time level. The purpose of solving the Riemann problem is simply to resolve the instantaneous wave interactions of the left and right states in order to obtain an upwind state at the cell edge.

2.1. Limited central difference approximations. The first step of the algorithm calculates $(\Delta U)_j$, a monotonized central difference approximation to $(\partial U/\partial x)\,\Delta x$ at each cell center:

$$(\Delta U)_j = \sum_{k=1}^{N} (\alpha_{k,j})\,(r_{k,j})\,.$$

The $r_{k,j}$ are the right eigenvectors of $A(U_j^n)$. The $\alpha_{k,j}$ are calculated as follows. Define three vectors, α^C, α^L, and α^R, by

$$\begin{aligned}\alpha^C &= L_j(\tfrac{1}{2}(U_{j+1} - U_{j-1})),\\ \alpha^L &= L_j(U_j - U_{j-1}),\\ \alpha^R &= L_j(U_{j+1} - Uj),\end{aligned}$$

where L_j is the matrix of left eigenvectors of $A(U_j^n)$. $\alpha_{k,j}$ is then given by

$$\begin{aligned}\alpha_{k,j} &= \min(|\alpha_k^C|, 2|\alpha_k^L|, 2|\alpha_k^R|) \times \operatorname{sgn}(\alpha_k^C) && \text{if } \alpha_k^L\alpha_k^R > 0,\\ &= 0 && \text{otherwise.}\end{aligned} \tag{2.3}$$

We call the values $\alpha_{k,j}$ *expansion coefficients*. In a given cell j, the expansion coefficients are the slopes of the linear profiles of the solution in the space of the right eigenvectors of $A(U_j^n)$. We refer to (2.3) as the *slope limiting procedure* because it generally sets $\alpha_{k,j} = \alpha_k^C$ unless there are nearby local extrema or large gradients in U.

Remark. The slope limiting procedure is an attempt to preserve monotonicity in each characteristic field [37]. Additional dissipation can be added near strong shocks by a slope-flattening algorithm [11], [14].

2.2. Characteristic tracing. The second step of the algorithm computes the time-centered left and right states at the cell edges, $U_{j+1/2,l}^{n+1/2}$ and $U_{j+1/2,r}^{n+1/2}$. The left and right states at a given cell edge are computed by second-order accurate Taylor expansions about the cell centers of the cells to the left and the right of the edge. In computing the left state, however, we use a characteristic projection operator that discards those components in the $\partial U/\partial x$ term corresponding to characteristics which do not originate in the cell to the left of the cell edge, i.e., only characteristics with positive speeds are used in the calculation of the left state. A similar procedure is followed for the right state. This procedure has no effect for problems in which $F(U) = AU$ where A is a constant matrix, since the discarded components do not affect the solution of the Riemann problem in that case. However, using the characteristic projection operators results in a more robust algorithm for strongly nonlinear problems [11].

We first look at the calculation of the left state. ($\lambda_{k,j}$, $l_{k,j}$, and $r_{k,j}$ are the kth eigenvalue, left eigenvector, and right eigenvector, respectively, of $A(U_j^n)$.) From Taylor's theorem,we have

$$\begin{aligned} U_{j+\frac{1}{2},l}^{n+\frac{1}{2}} &= U_j^n + \frac{\Delta x}{2}\left(\frac{\partial U}{\partial x}\right)_j^n + \frac{\Delta t}{2}\left(\frac{\partial U}{\partial t}\right)_j^n \\ &= U_j^n + \frac{\Delta x}{2}\left(\frac{\partial U}{\partial x}\right)_j^n + \frac{\Delta t}{2}\left(S\left(U_j^n, x\right) - \left(\frac{\partial F(U)}{\partial x}\right)_j^n\right) \\ &= U_j^n + \frac{1}{2}\left(I - \frac{\Delta t}{\Delta x}A\left(U_j^n\right)\right)(\Delta U)_j + \frac{\Delta t}{2}S\left(U_j^n, j\Delta x\right), \end{aligned}$$

where $(\cdot)_j^n$ denotes evaluation at $(t^n, j\Delta x)$. We now replace

$$\left(I - \frac{\Delta t}{\Delta x}A\left(U_j^n\right)\right)(\Delta U)_j$$

in the above expression by

$$P_L\left(\left(I - \frac{\Delta t}{\Delta x}A\left(U_j^n\right)\right)(\Delta U)_j\right),$$

where P_L is defined by

$$P_L(w) = \sum_{k:\lambda_{k,j}>0} (l_{k,j}\cdot w)\, r_{k,j}. \tag{2.4}$$

The resulting algorithm for calculating $U_{j+1/2,l}^{n+1/2}$ is

$$U_{j+\frac{1}{2},l}^{n+\frac{1}{2}} = U_j^n + \frac{1}{2}\sum_{k:\lambda_{k,j}>0}\left(1 - \frac{\Delta t}{\Delta x}\lambda_{k,j}\right)\alpha_{k,j} r_{k,j} + \frac{\Delta t}{2}S\left(U_j^n, j\Delta x\right).$$

Similarly, $U_{j+1/2,r}^{n+1/2}$ is given by

$$U_{j+\frac{1}{2},r}^{n+\frac{1}{2}} = U_{j+1}^n - \frac{1}{2}P_R\left(\left(I + \frac{\Delta t}{\Delta x}A\left(U_{j+1}^n\right)\right)(\Delta U)_{j+1}\right) + \frac{\Delta t}{2}S\left(U_{j+1}^n, (j+1)\Delta x\right),$$

where P_R is defined by

$$P_R(w) = \sum_{k:\lambda_{k,j+1}<0} (l_{k,j+1}\cdot w)\, r_{k,j+1}. \tag{2.5}$$

The resulting algorithm for calculating $U^{n+1/2}_{j+1/2,r}$ is

$$U^{n+\frac{1}{2}}_{j+\frac{1}{2},r} = U^n_{j+1} - \frac{1}{2} \sum_{k:\lambda_{k,j+1}<0} \left(1 + \frac{\Delta t}{\Delta x}\lambda_{k,j+1}\right) \alpha_{k,j+1} r_{k,j+1} + \frac{\Delta t}{2} S\left(U^n_{j+1}, (j+1)\Delta x\right).$$

Remark. The slope limiting procedure and the characteristic projection operators together guarantee that the following is true in the calculation of each left and right state at each edge: The origin of the characteristic in each characteristic field used in computing the value of the state is in the cell in which the linear profile corresponding to that field cannot contain new local extrema.

3. Second-order Godunov method for stiff relaxation. We now modify the nonstiff method to obtain a second-order Godunov method which computes higher-order accurate solutions of a stiff hyperbolic system of conservation laws with relaxation without fully resolving the effects of the stiff source terms. We refer to this method as the *frozen method* because its Riemann solver does not account for the equilibrium equation.

We consider a hyperbolic system of conservation laws with relaxation for which there is a single rate equation, i.e., a system of the form

$$(3.1)\qquad \begin{gathered} \frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = S(U,x) + H(U), \quad t \geq 0, \quad -\infty < x < \infty, \\ U = U(x,t), \quad F(U), \quad S(U,x), \quad H(U) \in \boldsymbol{R}^N, \end{gathered}$$

where

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = S(U,x)$$

is a hyperbolic conservation law with nonstiff source terms. $H(U)$ has the form $(0,0,\ldots,0,h(U))^T$. Define $M = N-1$ and U_M to be a vector of the first M components of U. We assume that a function $u_N = f(U_M)$ is defined implicitly by $h(U) = c$ for all values of c. The equilibrium value of u_N equals $u_*(U_M)$, where $u_*(U_M)$ satisfies the equation $h(U_M, u_*(U_M)) = 0$. We refer to (3.1) as the *nonequilibrium* system. The *equilibrium* system corresponding to (3.1) is

$$(3.2)\qquad \frac{U_M}{\partial t} + \frac{\partial F_*(U_M)}{\partial x} = S(U_M, x),$$

where $F_*(U_M) = F(U_M, u_*(U_M))$. We assume that the characteristic speeds of (3.1) and (3.2) are all distinct and that they satisfy the subcharacteristic condition (1.1) for all U_M. We also assume that all M characteristic fields of the equilibrium equation are either genuinely nonlinear or linearly degenerate. We finally assume that

$$\frac{\partial h(U)}{\partial(u_N - u_*(U_M))} < 0$$

for all U. The reciprocal of the absolute value of the left-hand side of this inequality is the relaxation time $\tau(U)$.

In our second-order Godunov method for (3.1), we want to use the time step given by (2.2) to advance the solution at time t_n even though the system may be stiff, i.e., $\max_{k,j}(\Delta x/\tau(U^n_j))/|\lambda^n_{k,j}| \gg 1$. We therefore make the algorithm in §2 semi-implicit with respect to $H(U)$ by modifying the second and fourth steps of the nonstiff method. We also make modifications to the characteristic projection operator used in the characteristic tracing step and to the limiting procedure (2.3) of the central difference approximation step. The first of these two changes ensures that the method is second-order accurate for stiff problems with smooth solutions. The second change helps the characteristic tracing step maintain some of the same properties as the tracing step in the nonstiff method.

The third step of the nonstiff method, the Riemann solver, does not have to be modified for the purpose of accuracy. On the other hand, the Riemann solver should satisfy (1.2) so that the overall method is an upwind-centered difference scheme for all nonnegative values of the relaxation time. At the present time, we believe that the development of a Riemann solver that satisfies (1.2) for a general hyperbolic conservation law with stiff relaxation is an open problem. Bolstad et al. [4] have made some progress in this area for a particular system. (See the comments in §1 for further discussion.) In the method presented here, we solve the same Riemann problem as in the nonstiff method in §2, i.e., assuming $H(U) \equiv 0$. In all our applications of the method in this paper, we use an exact Riemann solver.

In our presentation of the frozen method, we discuss the characteristic tracing step before the central difference approximation step in order to motivate the changes we make to the limiting procedure.

3.1. Characteristic tracing. We now modify the characteristic tracing step of the method in §2. This step requires two modifications. The first one is simply that the stiff source term $H(U)$ be treated semi-implicitly in the Taylor expansions, i.e., in the case of a left edge, H must be evaluated at $U^{n+1/2}_{j+1/2,l}$, not at U^n_j:

$$U^{n+\frac{1}{2}}_{j+\frac{1}{2},l} = U^n_j + \frac{1}{2}\left(I - \frac{\Delta t}{\Delta x}A\left(U^n_j\right)\right)(\Delta U)_j$$
$$+ \frac{\Delta t}{2}\left(S\left(U^n_j, j\Delta x\right) + H\left(U^{n+\frac{1}{2}}_{j+\frac{1}{2},l}\right)\right).$$

Similarly,

$$U^{n+\frac{1}{2}}_{j+\frac{1}{2},r} = U^n_{j+1} - \frac{1}{2}\left(I + \frac{\Delta t}{\Delta x}A\left(U^n_{j+1}\right)\right)(\Delta U)_{j+1}$$
$$+\frac{\Delta t}{2}\left(S\left(U^n_{j+1}, j\Delta x\right) + H\left(U^{n+\frac{1}{2}}_{j+\frac{1}{2},r}\right)\right).$$

The second modification is in the definitions of the characteristic projection operators P_L and P_R defined in (2.4) and (2.5). This change is needed to maintain second-order accuracy when the source term is stiff. The numerical method should essentially reduce to a second-order method for the equilibrium equation as $\tau(U) \to 0$. The method, then, should not "discard" any components of $\partial U/\partial x$ used by such a method. If the kth field of the equilibrium equation is needed for a second-order accurate solution of the equilibrium equation, the method should use the components of $\partial U/\partial x$ corresponding to the kth or the $(k+1)$st characteristic fields of (3.1) as $\tau(U) \to 0$, since disturbances carried by the kth field of the equilibrium equation must be present in the disturbances carried by the two fields of (3.1) [27], [43]. If the projection operators as defined in [11] are used, however, the method may "discard" some components of $\partial U/\partial x$ which are needed for second-order accuracy.

For example, suppose that the kth equilibrium characteristic speed in the jth cell is positive. The method should then use the kth and the $(k+1)$st characteristic fields of (3.1) in computing $U^{n+1/2}_{j+1/2,l}$. Since the subcharacteristic condition is satisfied, the component of $\partial U/\partial x$ corresponding to the $(k+1)$st characteristic field is not discarded by the projection operator P_L in the nonstiff method. However, the kth characteristic speed can be positive or negative. If it is negative, the component of $\partial U/\partial x$ corresponding to the kth characteristic field is discarded, and accuracy is lost as $\tau(U) \to 0$.

We therefore modify the characteristic projection operators in order to avoid a loss of accuracy as the relaxation time tends to zero. We use the following criteria so that a direct calculation of the equilibrium characteristic speeds is unnecessary:

(1) Use the kth characteristic field in the computation of $U^{n+1/2}_{j+1/2,l}$ if $k = N$ and $\lambda_{k,j} > 0$ or $k < N$ and $\lambda_{k+1,j} > 0$.

(2) Use the kth characteristic field in the computation of $U^{n+1/2}_{j+1/2,r}$ if $k = 1$ and $\lambda_{k,j+1} < 0$ or $k > 1$ and $\lambda_{k-1,j+1} < 0$.

These criteria are conservative in the sense that more components of $\partial U/\partial x$ may be kept than are necessary for second-order accuracy.

One result of using these criteria is that additional limiting may be needed in some of the expansion coefficients $\alpha_{k,j}$, i.e., the magnitude of $\alpha_{k,j}$ may have to be smaller than the magnitude calculated in (2.3). In order to minimize the additional amount of limiting, we find two sets of expansion coefficients in the central difference approximation step. We compute coefficients $\alpha^l_{k,j}$, which are used in the calculation of $U^{n+1/2}_{j+1/2,l}$, and $\alpha^r_{k,j}$, which are used in the calculation of $U^{n+1/2}_{j+1/2,r}$.

The resulting algorithm for calculating $U^{n+1/2}_{j+1/2,l}$ is the following:

$$\begin{aligned} U^{n+\frac{1}{2}}_{j+\frac{1}{2},l} &= U^n_j + \frac{1}{2} \sum_{k:\lambda_{k,j}>0 \text{ or } \lambda_{k+1,j}>0} \left(1 - \frac{\Delta t}{\Delta x}\lambda_{k,j}\right) \alpha^l_{k,j} r_{k,j} \\ &\quad + \frac{\Delta t}{2}\left(S\left(U^n_j, j\Delta x\right) + H\left(U^{n+\frac{1}{2}}_{j+\frac{1}{2},l}\right)\right). \end{aligned} \tag{3.3}$$

Similarly, the algorithm for $U^{n+1/2}_{j+1/2,r}$ is as follows:

$$\begin{aligned} U^{n+\frac{1}{2}}_{j+\frac{1}{2},r} &= U^n_{j+1} - \frac{1}{2} \sum_{k:\lambda_{k,j+1}<0 \text{ or } \lambda_{k-1,j+1}<0} \left(1 + \frac{\Delta t}{\Delta x}\lambda_{k,j+1}\right) \alpha^r_{k,j+1} r_{k,j+1} \\ &\quad + \frac{\Delta t}{2}\left(S\left(U^n_{j+1}, (j+1)\Delta x\right) + H\left(U^{n+\frac{1}{2}}_{j+\frac{1}{2},r}\right)\right). \end{aligned} \tag{3.4}$$

Both expressions are explicit for U_M but implicit for u_N. The Nth components of $U^{n+1/2}_{j+1/2,l}$ and $U^{n+1/2}_{j+1/2,r}$ are found by solving the equations defined by these expressions for u_N.

3.2. Limited central difference approximations. Our method for stiff relaxation may generate unacceptable oscillations in the numerical solution of a nonstiff strong shock problem if it uses the same central difference approximation step as the nonstiff method. This possibility arises because of the characteristic tracing algorithm, (3.3) and (3.4). Suppose, for example, that a characteristic field corresponding to a characteristic with a negative speed is used in the calculation of a left state at a cell edge. The origin of that characteristic is in the cell to the right of the cell edge. In that cell, the linear profile corresponding to that field which originates in the cell to the left of the edge may contain new local extrema; the slope-limiting procedure (2.3) prohibits the introduction of new local extrema only within the cell where the profile originates [36].

To eliminate the introduction of new extrema, our method uses a more restrictive slope-limiting procedure to calculate two sets of expansion coefficients. One set is used in the calculation of left edge states (3.3); the other is used in the calculation of right edge states (3.4). Although we could use a single set of expansion coefficients for both cell edges, we use two sets to minimize the dissipative effect [37] of the additional slope limiting. We now show the details of calculating $\alpha^l_{k,j}$, the expansion coefficient for the kth characteristic field used in calculating $U^{n+1/2}_{j+1/2,l}$; the details of the calculation of the expansion coefficients used in calculating $U^{n+1/2}_{j+1/2,r}$ are similar.

α^C, α^L, and α^R are defined as in §2.1. If $\lambda_{k,j} > 0$, (2.3) is used to find $\alpha^l_{k,j}$. However, if $\lambda_{k,j} \le 0$, $\alpha^l_{k,j}$ is given by the following:

$$\begin{aligned} \alpha^l_{k,j} &= \min\left(\left|\alpha^C_k\right|, 2\left|\alpha^L_k\right|, \beta_{k,j}\left|\alpha^R_k\right|\right) \times \operatorname{sgn}\left(\alpha^C_k\right) && \text{if } \alpha^L_k\alpha^R_k > 0, \\ &= 0 && \text{otherwise,} \end{aligned} \tag{3.5}$$

where

$$\beta_{k,j} = 2 - \left|\lambda_{k,j}\right| \frac{\Delta t}{\Delta x}. \tag{3.6}$$

The CFL condition implies that $2 - \sigma \le \beta \le 2$. Since β is bounded away from 1, our method is still second-order accurate. (See van Leer [37] for further discussion.) Suppose $\lambda_{k,j} \le 0$ and the kth characteristic field is used in the calculation of $U^{n+1/2}_{j+1/2,l}$. The value of $\beta_{k,j}$ calculated in (3.6) is then the largest possible value that can be used in (3.5) which guarantees the following: if the kth characteristic field is used in the calculation of $U^{n+1/2}_{j+1/2,l}$, the linear profile corresponding to that field does not contain any value outside the range of values determined by cells j and $j+1$.

3.3. Conservative differencing. We now modify the fourth step of the method in §2. Suppose we wanted to obtain a globally second-order accurate numerical solution to the stiff *ordinary* differential equation

$$\frac{du_N}{dt} = h(U_M, u_N) \tag{3.7}$$

using a time step $\Delta t \gg \tau(U)$ where U_M is fixed. We want the numerical solution of u_N to decay rapidly to $u_*(U_M)$ as $\tau(U) \to 0$. Hence, we require a second-order method that is L-stable. An L-stable method is one for which numerical solutions of the test equation $y' = \lambda y$ decay rapidly to zero as $\lambda \to -\infty$. (See Lambert [24] for a detailed discussion of numerical methods for stiff ordinary differential equations.)

We see from this discussion that ideally we want to incorporate a second-order, single-step, L-stable method into the fourth step of the Godunov method to account for the contribution of $H(U)$. The condition that the method be single step requires us to use an L-stable, implicit R-stage Runge–Kutta method, $R \ge 2$. These methods, however, can be computationally expensive since each stage requires the solution of a nonlinear equation. Yee [45] discusses alternatives to using a second-order, single-step, L-stable method.

In this paper we defer the issue of a second-order L-stable method because our main concern is with maintaining second-order accuracy as the relaxation time approaches zero. Hence, for the purpose of this paper, it is sufficient to use any L-stable method, since any such method would set $u_{N,j}^{n+1}$ to $u_*(U_{M,j}^{n+1})$ in the limit of $\tau \to 0$. The method used to calculate the numerical results in this paper has a conservative differencing step which incorporates the backward Euler method, a first-order L-stable method:

$$(3.8)\quad \begin{aligned} U_j^{n+1} &= U_j^n + \frac{\Delta t}{\Delta x}\left(F_{j-\frac{1}{2}}^{n+\frac{1}{2}} - F_{j+\frac{1}{2}}^{n+\frac{1}{2}}\right) + \Delta t H\left(U_j^{n+1}\right) \\ &\quad + \frac{\Delta t}{2}\left(S\left(U_{j-\frac{1}{2}}^{n+\frac{1}{2}}, \left(j-\frac{1}{2}\right)\Delta x\right) + S\left(U_{j+\frac{1}{2}}^{n+\frac{1}{2}}, \left(j+\frac{1}{2}\right)\Delta x\right)\right). \end{aligned}$$

This expression is explicit for U_M but implicit for u_N. u_N is updated by solving the equation for $u_{N,j}^{n+1}$ defined by (3.8).

We note that the method used in calculating the results in this paper is second-order accurate only in the limit $\tau(U) \to 0$. For long relaxation times, the method is only first-order accurate because a second-order L-stable method is not incorporated into its conservative differencing step.

4. Three variant methods. We now briefly present three variants of the method in §3. These methods should not be considered potential alternatives to the method in the previous section. We introduce these variants solely to help us analyze and illustrate through numerical results certain features of the frozen method.

The first variant is a fractional step method which uses Strang splitting. In the first and third fractional steps, a second-order L-stable method is used to integrate

$$\frac{\partial U}{\partial t} = H(U)$$

for time steps of $\Delta t/2$. In the second step, the second-order Godunov method for nonstiff source terms is used to integrate

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = S(U, x)$$

for a time step of Δt. The time step is governed by the CFL condition (2.2). We will refer to this variant as the *fractional step method.* We will use the method to demonstrate that an operator split treatment of the stiff source term is insufficient for obtaining higher-order accuracy.

The second variant is nearly identical to the method presented in the previous section. The only difference is that the characteristic projection operators P_L and P_R are identical to the ones used in the nonstiff method. We will refer to this variant as the *nonstiff projection method.* We will use the method to demonstrate that the characteristic projection operators in the frozen method must differ from the operators of the nonstiff method to obtain higher-order accuracy.

The third variant is also nearly identical to the method presented in the previous section. The only difference is that a Riemann solver for the equilibrium equation (3.2) is used in place of a Riemann solver for (3.1) to find the first M components of $U_{j+1/2}^{n+1/2}$; the nth component of $U_{j+1/2}^{n+1/2}$ is set to $u_*((U_M)_{j+1/2}^{n+1/2})$. We note that this variant is strictly applicable only in the limit $\tau(U) \to 0$. We will refer to this variant as the *equilibrium method*. We will use the method to demonstrate that the method in §3 may be improved if its Riemann solver satisfies (1.2).

5. Formulation of model system and test problems. In this section, we formulate a model hyperbolic system of conservation laws with relaxation. We then formulate two specific test problems. The model system is derived from the system in [31] by applying the following change of coordinates to that system:

$$x \leftarrow X + aT,$$
$$t \leftarrow T.$$

Our model is an example of the system examined by Liu [27]. It is also similar to the combustion model considered by Majda [29]; see the discussion in [31].

5.1. A model system for relaxation. We consider the following system of hyperbolic conservation laws with relaxation:

$$\frac{\partial}{\partial t}\begin{pmatrix} w \\ z \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} \frac{1}{2}u^2 + aw \\ az \end{pmatrix} = \begin{pmatrix} 0 \\ -K(z - m(u - u_0)) \end{pmatrix}, \tag{5.1}$$

where $u = w - q_0 z$, and a, K, q_0, m, and u_0 are constants satisfying $K > 0$, $q_0 < 0$, $m < 0$, and $u_0 > 0$. a can be zero, positive, or negative. The relaxation time of the system is $1/K$. The system has characteristic speeds $\lambda_1(w, z) = a$ and $\lambda_2(w, z) = u + a$. We consider only those problems for which $u > 0$ for all x, t.

The equilibrium equation for (5.1) is

$$\frac{\partial w}{\partial t} + \frac{\partial}{\partial x}\left(\frac{1}{2}u_*(w)^2 + aw\right) = 0, \tag{5.2}$$

where $u_*(w)$ is defined by

$$\begin{aligned} u_*(w) &= w - q_0 z_*(w), \\ z_*(w) &= \frac{m}{1 + mq_0}(w - u_0). \end{aligned} \tag{5.3}$$

$z_*(w)$ is the equilibrium value of z for a given value of w. The equilibrium characteristic speed λ_* is

$$\lambda_*(w) = u_*(w)u_*'(w) + a = \frac{u_*(w)}{1 + mq_0} + a. \tag{5.4}$$

We observe that when $u > 0$, the frozen and the equilibrium characteristics satisfy the subcharacteristic condition.

5.2. Two test problems. We now formulate and solve two problems of the model system. Both problems consist of finding the limit as $K \to \infty$ of the solution of an initial value problem of (5.1). The problems are formulated so that the initial values are in equilibrium and $u_*(w) > 0$ for all x, t. We assume the results of Liu [28] and calculate these limits indirectly by finding solutions of the equilibrium equation. The solution of the first problem is a rarefaction wave of the equilibrium equation; the solution of the second is a shock wave.

The first test problem is to find $\lim_{K\to\infty} w(x,t)$ where

$$\begin{aligned}(w,z)(x,0) &= (w_l, z_*(w_l)), \qquad x < x_0,\\ &= (w_r, z_*(w_r)), \qquad x > x_0,\end{aligned}$$

and $0 < u_l = w_l - q_0 z_*(w_l) < u_r = w_r - q_0 z_*(w_r)$. The solution of this problem is

$$\begin{aligned}\lim_{K\to\infty} w(x,t) &= w_l, \qquad \frac{x-x_0}{t} \le \frac{u_l}{1+q_0 m} + a,\\ &= w_r, \qquad \frac{x-x_0}{t} \ge \frac{u_r}{1+q_0 m} + a,\\ &= \frac{x-x_0}{t}, \qquad \text{otherwise.}\end{aligned}$$

We will refer to this problem as the rarefaction test problem.

The second test problem is to find $\lim_{K\to\infty} w(x,t)$ where

$$\begin{aligned}(w,z)(x,0) &= (w_l, z_*(w_l)), \qquad x < x_0,\\ &= (w_r, z_*(w_r)), \qquad x > x_0,\end{aligned}$$

and $u_l = w_l - q_0 z_*(w_l) > u_r = w_r - q_0 z_*(w_r) > 0$. The solution of this problem is

$$\begin{aligned}\lim_{K\to\infty} w(x,t) &= w_l, \qquad x < x_0 + st,\\ &= w_r, \qquad x > x_0 + st,\end{aligned}$$

where $s = \frac{1}{2}(u_l^2 - u_r^2)/(w_l - w_r)$. We will refer to this problem as the shock test problem.

6. Analysis of numerical methods for stiff problems. In this section we analyze how well the methods in §3 and the variant methods in §4 approximate solutions of the model equations (5.1) as $K \to \infty$. There are two main goals of this analysis. The first is to show that two of the features of the method in §3, the coupling of the source term in the Godunov method and the modified characteristic tracing step are, in some sense, necessary features of a second-order Godunov method for stiff relaxation. The other goal is to show that the method in §3 may be more robust for stiff problems if the Riemann solver is modified so that the solver satisfies the property in (1.2). We accomplish the first goal by showing that the method in §3 is a second-order accurate method while the fractional step method and the nonstiff projection method in §4 are first-order accurate methods. The second goal is achieved by showing that the method in §3 does not reduce to an upwind method for the equilibrium equation as the relaxation time tends to zero. The equilibrium method, on the other hand, is an upwind method for the equilibrium equation by construction.

The analysis in this section is based on an observation and an assumption. The observation is that for the model equations in §5, the method in §3 and the three variants in §4 can be expressed in the following form as $K \to \infty$:

$$w_j^{n+1} = w_j^n + \frac{\Delta t}{\Delta x}\left(f_{w,j-\frac{1}{2}}^{n+\frac{1}{2}} - f_{w,j+\frac{1}{2}}^{n+\frac{1}{2}}\right),$$

$$z_j^{n+1} = z_*\left(w_j^{n+1}\right),$$

$$f_{w,j+\frac{1}{2}}^{n+\frac{1}{2}} = \frac{1}{2}\left(w_{j+\frac{1}{2}}^{n+\frac{1}{2}} - q_0 z_{j+\frac{1}{2}}^{n+\frac{1}{2}}\right)^2 + a w_{j+\frac{1}{2}}^{n+\frac{1}{2}}.$$

The only difference among the methods in this limit is in the calculations of $w_{j+1/2}^{n+1/2}$ and $z_{j+1/2}^{n+1/2}$. The assumption is that Liu's conjecture [28] is true for the model system, i.e., solutions of (5.1) tend to solutions of (5.2) as $K \to \infty$. The analysis then reduces to examining how well the four methods calculate solutions of the equilibrium equation in the limit $K \to \infty$.

In the analysis of accuracy we use the fact that all four methods are upwind-centered difference schemes. Any one of these methods, then, is a globally second-order accurate method (i.e., a locally third-order accurate method) in the limit $K \to \infty$ if and only if the local truncation errors in the calculation of $w_{j+1/2}^{n+1/2}$ and $z_{j+1/2}^{n+1/2}$ are $O(\Delta x^2)$, i.e.,

$$
\begin{aligned}
w_{j+\frac12}^{n+\frac12} &= w_j^n + \frac{\Delta x}{2}\left(\frac{\partial w}{\partial x}\right)_j^n + \frac{\Delta t}{2}\left(\frac{\partial w}{\partial t}\right)_j^n + O\left(\Delta x^2\right) \\
&= w_j^n + \left(\frac{\Delta x}{2} - \frac{\Delta t}{2}\left(u_*\left(w_j^n\right)u_*'\left(w_j^n\right) + a\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right)
\end{aligned} \tag{6.1}
$$

and

$$
\begin{aligned}
z_{j+\frac12}^{n+\frac12} &= z_*\left(w_j^n + \frac{\Delta x}{2}\left(\frac{\partial w}{\partial x}\right)_j^n + \frac{\Delta t}{2}\left(\frac{\partial w}{\partial t}\right)_j^n\right) + O\left(\Delta x^2\right) \\
&= z_*\left(w_j^n\right) \\
&\quad + z_*'\left(w_j^n\right)\frac{1}{2}\left(\Delta x - \Delta t\left(u_*\left(w_j^n\right)u_*'\left(w_j^n\right) + a\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right),
\end{aligned} \tag{6.2}
$$

where $(\cdot)_j^n$ denotes evaluation at t_n, $j\Delta x$.

Most of this analysis consists of examining the truncation error of the characteristic tracing step in each method. We assume that $\Delta w/\Delta x$ and $\Delta z/\Delta x$ are first-order approximations to $\partial w/\partial x$ and $\partial z/\partial x$, i.e., we do not attempt to include the effect of slope limiting, (2.3) or (3.5).

6.1. Characteristic tracing: Fractional step method. We first demonstrate that an unsplit methodology is necessary for obtaining second-order accurate solutions of the model system as $K \to \infty$. We verify this by showing that the Godunov step in the fractional step method in §4 is only first-order accurate in this limit.

It is sufficient to consider the case $a > 0$. When $a > 0$, the Riemann solver sets $z_{j+1/2}^{n+1/2} = z_{j+1/2,l}^{n+1/2}$. We expand in Taylor's series the terms of the expressions used in the characteristic tracing step to calculate $z_{j+1/2,l}^{n+1/2}$ as follows:

$$
\begin{aligned}
z_{j+\frac12,l}^{n+\frac12} &= z_j^n + \frac12\left(1 - \frac{\Delta t}{\Delta x}a\right)(\Delta z)_j^n \\
&= z_*\left(w_j^n\right) + \frac12\left(\Delta x - \Delta t a\right)\left(\frac{\partial z}{\partial x}\right)_j^n + O\left(\Delta x^2\right) \\
&= z_*\left(w_j^n\right) + \frac12\left(\Delta x - \Delta t a\right)z_*'\left(w_j^n\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right).
\end{aligned}
$$

Since the difference between (6.2) and the last expression on the right is $O(\Delta x)$, the fractional step method is only first-order accurate.

6.2. Characteristic tracing: Nonstiff projection method. We demonstrate that the characteristic projection operators in the tracing step of the frozen method must differ from those operators in the nonstiff method. The characteristic projection operators must be different in order for the frozen method to produce second-order accurate solutions of the model system as $K \to \infty$. We verify this by showing that the nonstiff projection method in §4 is only first-order accurate as $K \to \infty$ when the frozen and the equilibrium characteristic speeds of the model system do not have the same sign.

It is sufficient to consider the case $a < 0 < u + a$ for all u under consideration. When $a < 0 < u + a$ for all u, the Riemann solver calculates $w_{j+1/2}^{n+1/2}$ as follows:

$$w_{j+\frac{1}{2}}^{n+\frac{1}{2}} = w_{j+\frac{1}{2},l}^{n+\frac{1}{2}} - q_0 z_* \left(w_{j+\frac{1}{2},l}^{n+\frac{1}{2}}\right) + q_0 z_* \left(w_{j+\frac{1}{2},r}^{n+\frac{1}{2}}\right). \tag{6.3}$$

We expand in Taylor's series the terms of the expressions used in the characteristic tracing step to calculate $w_{j+1/2,l}^{n+1/2}$, $w_{j+1/2,r}^{n+1/2}$, $z_{j+1/2,l}^{n+1/2}$, and $z_{j+1/2,r}^{n+1/2}$ as follows:

$$\begin{aligned}
w_{j+\frac{1}{2},l}^{n+\frac{1}{2}} &= w_j^n + \frac{1}{2}\left(\Delta x - \left(u_*\left(w_j^n\right) + a\right)\Delta t\right)\left(\Delta w_j^n - q_0 \Delta z_j^n\right), \\
&= w_j^n + \frac{1}{2}\left(\Delta x - \Delta t\left(u_*\left(w_j^n\right) u_*'\left(w_j^n\right) + a\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right), \\
w_{j+\frac{1}{2},r}^{n+\frac{1}{2}} &= w_{j+1}^n - \frac{1}{2}\left(\Delta x + \left(u_*\left(w_j^n\right) + a\right)\Delta t\right)\left(\Delta w_{j+1}^n - q_0 \Delta z_{j+1}^n\right) \\
&= w_j^n + \left(\Delta x - \frac{1}{2}\left(\Delta x + \Delta t a\right)\left(1 - u_*'\left(w_j^n\right)\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right), \\
z_{j+\frac{1}{2},l}^{n+\frac{1}{2}} &= z_*\left(w_{j+\frac{1}{2},l}^{n+\frac{1}{2}}\right) \\
&= z_*\left(w_j^n\right) + z_*'\left(w_j^n\right)\left(\frac{1}{2}\left(\Delta x - \Delta t\left(u_*\left(w_j^n\right) u_*'\left(w_j^n\right) + a\right)\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n \\
&\quad + O\left(\Delta x^2\right), \\
z_{j+\frac{1}{2},r}^{n+\frac{1}{2}} &= z_*\left(w_{j+\frac{1}{2},r}^{n+\frac{1}{2}}\right) \\
&= z_*\left(w_j^n\right) + z_*'\left(w_j^n\right)\left(\Delta x - \frac{1}{2}\left(\Delta x + \Delta t a\right)\left(1 - u_*'\left(w_j^n\right)\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n \\
&\quad + O\left(\Delta x^2\right).
\end{aligned}$$

After substituting the above expressions into (6.3), we obtain after some manipulations the following equation:

$$\begin{aligned}
w_{j+\frac{1}{2}}^{n+\frac{1}{2}} = w_j^n + \Bigg(\frac{\Delta x}{2} - \frac{\Delta t}{2}\left(u_*\left(w_j^n\right) u_*'\left(w_j^n\right) + a\right) + q_0 z_*'\left(w_j^n\right) u_*'\left(w_j^n\right) \\
\left(\Delta x - u_*\left(w_j^n\right)\Delta t\right)\Bigg)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right).
\end{aligned}$$

Since the difference between the expression on the right and (6.1) is $O(\Delta x)$, the nonstiff projection method is only first-order accurate.

Remark. We emphasize that the first-order accuracy of the nonstiff projection method does not depend in any way on the order of the L-stable method incorporated into the conservative differencing step. In the analysis above, we examined the nonstiff projection method in the limit $K \to \infty$. In this limit, the method performs in the same manner regardless of the choice of L-stable scheme.

6.3. Characteristic tracing: Frozen method. We now show that the method in §3 is second order accurate as $K \to \infty$. The value of $z_{j+1/2}^{n+1/2}$ is either $z_*(w_{j+1/2,l}^{n+1/2})$ or $z_*(w_{j+1/2,r}^{n+1/2})$, depending on the sign of a. The value of $w_{j+1/2}^{n+1/2}$ calculated by the Riemann solver is some linear combination of the values of $w_{j+1/2,l}^{n+1/2}$, $w_{j+1/2,r}^{n+1/2}$, $z_*(w_{j+1/2}^{n+1/2}, l)$, and $z_*(w_{j+1/2,r}^{n+1/2})$. It is sufficient, then, to show that the truncation errors in the calculations of $w_{j+1/2,l}^{n+1/2}$ and $w_{j+1/2,r}^{n+1/2}$ are $O(\Delta x^2)$.

We expand in Taylor's series the terms of the expressions used in the characteristic tracing step to calculate $w_{j+1/2,l}^{n+1/2}$ and $w_{j+1/2,r}^{n+1/2}$ as follows:

$$\begin{aligned}
w_{j+\frac{1}{2},l}^{n+\frac{1}{2}} &= w_j^n + \frac{1}{2}\left(1 - \frac{\Delta t}{\Delta x}\left(u_*\left(w_j^n\right) + a\right)\right)\left(\Delta w - q_0 \Delta z\right) \\
&\quad + \frac{1}{2}\left(1 - a\frac{\Delta t}{\Delta x}\right) q_0 \Delta z \\
&= w_j^n + \frac{1}{2}\left(\Delta x - \Delta t\left(u_*\left(w_j^n\right) + a\right)\right) u_*'\left(w_j^n\right)\left(\frac{\partial w}{\partial x}\right)_j^n \\
&\quad + \left(\Delta x - a\Delta t\right) q_0 z_*'\left(w_j^n\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right) \\
&= w_j^n + \frac{1}{2}\left(\Delta x - \Delta t\left(u_*\left(w_j^n\right) u_*'\left(w_j^n\right) + a\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right), \\
w_{j+\frac{1}{2},r}^{n+\frac{1}{2}} &= w_{j+1}^n - \frac{1}{2}\left(1 + \frac{\Delta t}{\Delta x}\left(u_*\left(w_j^n\right) + a\right)\right)\left(\Delta w - q_0 \Delta z\right) \\
&\quad - \frac{1}{2}\left(1 + a\frac{\Delta t}{\Delta x}\right) q_0 \Delta z \\
&= w_{j+1}^n - \frac{1}{2}\left(\Delta x + \Delta t\left(u_*\left(w_{j+1}^n\right) + a\right)\right) u_*'\left(w_{j+1}^n\right)\left(\frac{\partial w}{\partial x}\right)_{j+1}^n + O\left(\Delta x^2\right) \\
&\quad + \left(\Delta x + a\Delta t\right) q_0 z_*'\left(w_{j+1}^n\right)\left(\frac{\partial w}{\partial x}\right)_{j+1}^n \\
&= w_{j+1}^n - \frac{1}{2}\left(\Delta x + \Delta t\left(u_*\left(w_{j+1}^n\right) u_*'\left(w_{j+1}^n\right) + a\right)\right)\left(\frac{\partial w}{\partial x}\right)_{j+1}^n + O\left(\Delta x^2\right) \\
&= w_j^n + \frac{1}{2}\left(\Delta x - \Delta t\left(u_*\left(w_j^n\right) u_*'\left(w_j^n\right) + a\right)\right)\left(\frac{\partial w}{\partial x}\right)_j^n + O\left(\Delta x^2\right).
\end{aligned}$$

Hence, the method is second-order accurate in the limit $K \to \infty$.

6.4. Riemann solvers: Frozen vs. equilibrium. We now compare the Riemann solvers in the frozen method in §3 and in the equilibrium method in §4 in order to show that the frozen method does not reduce to an upwind scheme for the equilibrium equation (5.2) as $K \to \infty$. The following are necessary conditions for either method to be an upwind scheme [21] for (5.2) in this limit:

(1) If $\lambda_*(w^{n+1/2}_{j+1/2,l}) < 0$ and $\lambda_*(w^{n+1/2}_{j+1/2,r}) < 0$, then

$$w^{n+\frac{1}{2}}_{j+\frac{1}{2}} = w^{n+\frac{1}{2}}_{j+\frac{1}{2},r} \quad \text{and} \quad z^{n+\frac{1}{2}}_{j+\frac{1}{2}} = z^{n+\frac{1}{2}}_{j+\frac{1}{2},r} = z_*(w^{n+\frac{1}{2}}_{j+\frac{1}{2},r}).$$

(2) If $\lambda_*(w^{n+1/2}_{j+1/2,l}) > 0$ and $\lambda_*(w^{n+1/2}_{j+1/2,r}) > 0$, then

$$w^{n+\frac{1}{2}}_{j+\frac{1}{2}} = w^{n+\frac{1}{2}}_{j+\frac{1}{2},l} \quad \text{and} \quad z^{n+\frac{1}{2}}_{j+\frac{1}{2}} = z^{n+\frac{1}{2}}_{j+\frac{1}{2},l} = z_*(w^{n+\frac{1}{2}}_{j+\frac{1}{2},l}).$$

The Riemann solver for the method in §3, which we will call the frozen Riemann solver, solves the Riemann problem for the system

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2} + au\right) = 0,$$

$$\frac{\partial z}{\partial t} + \frac{\partial}{\partial x}(az) = 0.$$

(This system is derived from the system

$$\frac{\partial w}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2} + aw\right) = 0,$$

$$\frac{\partial z}{\partial t} + \frac{\partial}{\partial x}(az) = 0$$

by using $w = u + q_0 z$ and substituting the second equation into the first.) The equilibrium Riemann solver solves the Riemann problem for the equilibrium equation (5.2). Both Riemann solvers calculate the same values of $w^{n+1/2}_{j+1/2}$ and $z^{n+1/2}_{j+1/2}$ if the frozen and the equilibrium characteristic speeds of the left and right states at a cell edge all have the same sign. However, if the characteristic speeds are not all of the same sign, the two solvers may calculate different values of $w^{n+1/2}_{j+1/2}$ and $z^{n+1/2}_{j+1/2}$.

As an example, we consider the following situation. If

$$a < u^{n+\frac{1}{2}}_{j+\frac{1}{2},l}/(1 + mq_0) + a < 0 < u^{n+\frac{1}{2}}_{j+\frac{1}{2},l} + a,$$

$$a < u^{n+\frac{1}{2}}_{j+\frac{1}{2},r}/(1 + mq_0) + a < 0 < u^{n+\frac{1}{2}}_{j+\frac{1}{2},r} + a,$$

$$u^{n+\frac{1}{2}}_{j+\frac{1}{2},l} > u^{n+\frac{1}{2}}_{j+\frac{1}{2},r},$$

both Riemann solvers find that the solution of the Riemann problem contains a shock separating $u^{n+1/2}_{j+1/2,l}$ and $u^{n+1/2}_{j+1/2,r}$. However, the frozen Riemann solver calculates a positive shock speed and sets $u^{n+1/2}_{j+1/2} = u^{n+1/2}_{j+1/2,l}$, while the equilibrium Riemann solver calculates a negative shock speed and sets $u^{n+1/2}_{j+1/2} = u^{n+1/2}_{j+1/2,r}$. Consequently, the method in §3 does not reduce to an upwind method for the equilibrium equation. As a result, the method in §3 is more likely than the equilibrium method to generate numerical oscillations or overshoots at shocks [35] as $K \to \infty$ when solving problems in which the frozen and the equilibrium characteristic speeds do not all have the same sign. By the same reasoning, however, the equilibrium method is more likely to generate oscillations for nonstiff problems. The equilibrium method is thus not suitable for long relaxation times.

7. Numerical results: Model system. In this section we use the numerical method developed in §3 and the three variant methods presented in §4 to solve examples of the test problems formulated in §5 in order to verify the analysis of the methods in §6. We look at three sets of numerical solutions of rarefaction and shock test problems. The first set of solutions are generated by the fractional step method in §4 and by the frozen method in §3. The results show that the fractional step method produces less accurate solutions than the frozen method as $K \to \infty$. The second set is generated by the nonstiff projection method in §4 and by the frozen method. The results show that the nonstiff projection method also produces less accurate solutions for stiff problems. The last set of solutions is generated by the frozen method. The results are consistent with the method being second-order accurate for smooth solutions as $K \to \infty$. Furthermore, the results the frozen method for the shock problems in which the frozen and equilibrium characteristic speeds are not all of the same sign contain no oscillations or overshoots at the shocks. Hence, there is no degradation in the numerical solution due to the fact that the Riemann solver does not satisfy (1.2).

In every problem discussed in this section, we define the parameters (except a) of the model system in §5 as follows:

$$\begin{aligned} q_0 &= -1.0. \\ m &= -1.0, \\ u_0 &= 3.0, \\ K &= 10^8. \end{aligned}$$

The value of a is set in each problem; we use different values in different problems in order to illustrate certain numerical properties of the methods. We define the initial conditions of the rarefaction problems by

$$\begin{aligned} u_l &= 2.0, \\ z_l &= m(u_l - u_0) = 1.0, \\ w_l &= u_l + q_0 z_l = 1.0, \\ u_r &= 3.0, \\ z_r &= m(u_r - u_0) = 0.0, \\ w_r &= u_r + q_0 z_r = 2.0. \end{aligned}$$

We define the initial conditions of the shock problems by reversing the roles of the left and right states. In each rarefaction problem, we set the position of the initial discontinuity, x_0, according to the value of a so that the solutions of all the rarefaction problems are identical at $t = .3$:

$$\begin{aligned} u(x, t = .3) &= 2, && x \leq .7, \\ &= 2 + (x - .7)/(.85 - .7), && .7 < x < .85, \\ &= 3, && x \geq .85. \end{aligned} \tag{7.1}$$

In each shock problem, we set x_0 in a similar fashion so that

$$\begin{aligned} u(x, t = .3) &= 3, && x < .775, \\ &= 2, && x > .775. \end{aligned} \tag{7.2}$$

We use a grid spacing of $\Delta x = .0025$ and a Courant number of $\sigma = .9$ in each run. We also use the following measure of error so that we can compare the numerical results of the different methods:

$$\text{error} = \frac{1}{m} \sum_{j=1}^{m} \left| u\text{eq}(x_j, t_n) - u_j^n \right|. \tag{7.3}$$

$u_{\text{eq}}(x_j, t_n)$ is the exact solution of the equilibrium equation (5.2) given by (7.1) or (7.2), while u_j^n is the numerical approximation to the solution at (x_j, t_n). m is the number of computational cells.

7.1. Frozen method vs. fractional step method. In this set of problems, we set $a = 1.0$. We use the fractional step method in §4 and the method in §3 to solve a shock test problem and a rarefaction test problem. The values of the numerical solutions at $t = .3$ are displayed in the graphs in Fig. 1. We can see from the figure that the results generated by the fractional step method are less accurate than those generated by the frozen method. Specifically, in the numerical solutions produced by the fractional step method, the representations of the trailing and leading edges of the rarefaction and the representation of the shock are more smeared than those in the solutions produced by the frozen method. The errors at $t = .3$, as given by (7.3), in the shock and the rarefaction solutions calculated by the frozen method are 1.108×10^{-3} and 1.104×10^{-3}. The corresponding values for the fractional step method are 1.931×10^{-3} and 4.149×10^{-3}.

7.2. Frozen method vs. nonstiff projection method. In this set of problems, we set $a = -.9$ so that the frozen and the equilibrium characteristic speeds do not all have the same sign. We expect in this case that the numerical solutions generated by the nonstiff projection method will be less accurate. We use the nonstiff projection method in §4 and the frozen method in §3 to solve a shock test problem and a rarefaction test problem. The values of the numerical solutions at $t = .3$ are displayed in the graphs in Fig. 2. We can see from the figure that the results generated by the nonstiff projection method are less accurate than those generated by the frozen method. Specifically, we see that the representations of the trailing and leading edges of the rarefaction and the representation of the shock are more smeared in the solutions produced by the nonstiff projection method than in those produced by the frozen method. The errors at $t = .3$, as given by (7.3), in the shock and the rarefaction solutions calculated by the frozen method are 1.170×10^{-3} and 1.117×10^{-3}. The corresponding values for the nonstiff projection method are 6.684×10^{-3} and 1.291×10^{-2}.

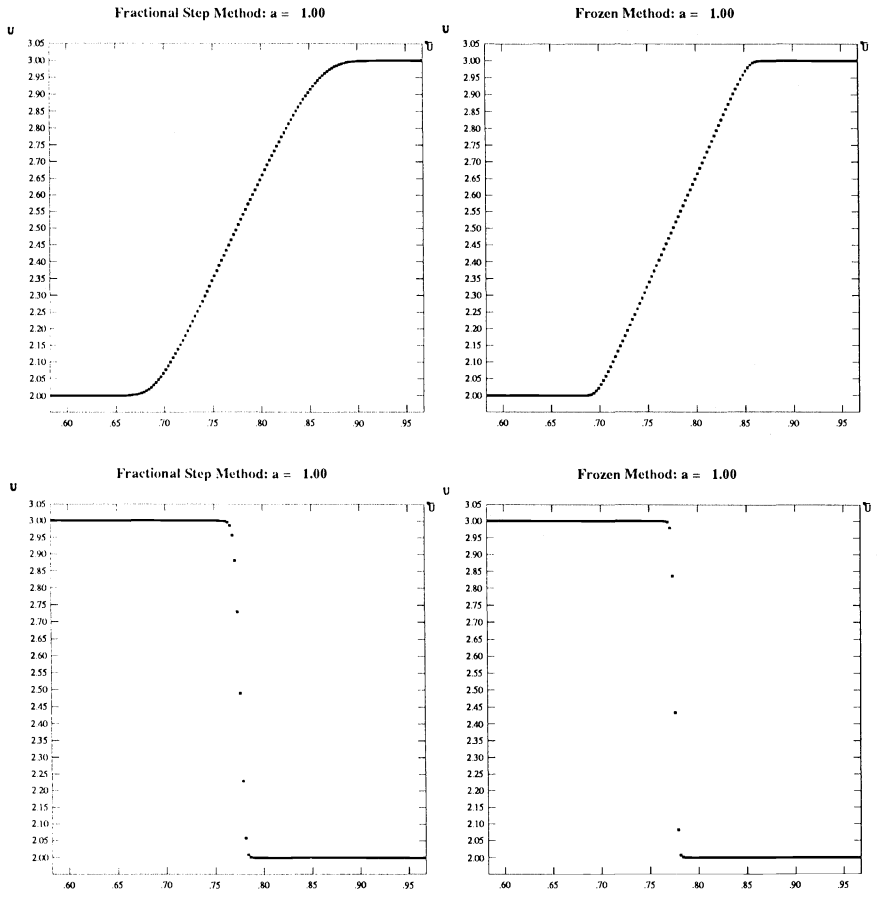


FIG. 1. *Numerical solutions of test shock and rarefaction problems by the fractional step and frozen methods for* $a = 1$.

7.3. Frozen method. In this set of problems, we set $a = -3.25, -2.5, -1.75, -1.25, -.9$, and 1.0. For each value of a we use the method in §3 to solve a shock test problem and a rarefaction test problem. The resulting twelve problems correspond to twelve different relations among the frozen and the equilibrium characteristic speeds of the left and right states. For example, when $a = -3.25$ the frozen and equilibrium characteristic speeds of the left and right states are all negative. The values of the numerical solutions for the first four values of a at $t = .3$ are displayed in the graphs in Figs. 3 and 4. The results for $a = -.9$ and for $a = 1.0$ are displayed in Fig. 2 and Fig. 1, respectively. The results are consistent with the method being second-order accurate, i.e., the frozen method produces sharply defined equilibrium shock and rarefaction solutions for all six values of a. The errors at $t = .3$, as given by (7.3), in the solutions to the six shock problems are, in order of increasing a, 1.069×10^{-3}, 1.046×10^{-3}, 1.182×10^{-3}, 1.213×10^{-3}, 1.170×10^{-3}, and 1.108×10^{-3}. The corresponding values for the six rarefaction problems are, in order of increasing a, 1.058×10^{-3}, 9.671×10^{-4}, 1.099×10^{-3}, 1.138×10^{-3}, 1.117×10^{-3}, and 1.104×10^{-3}.

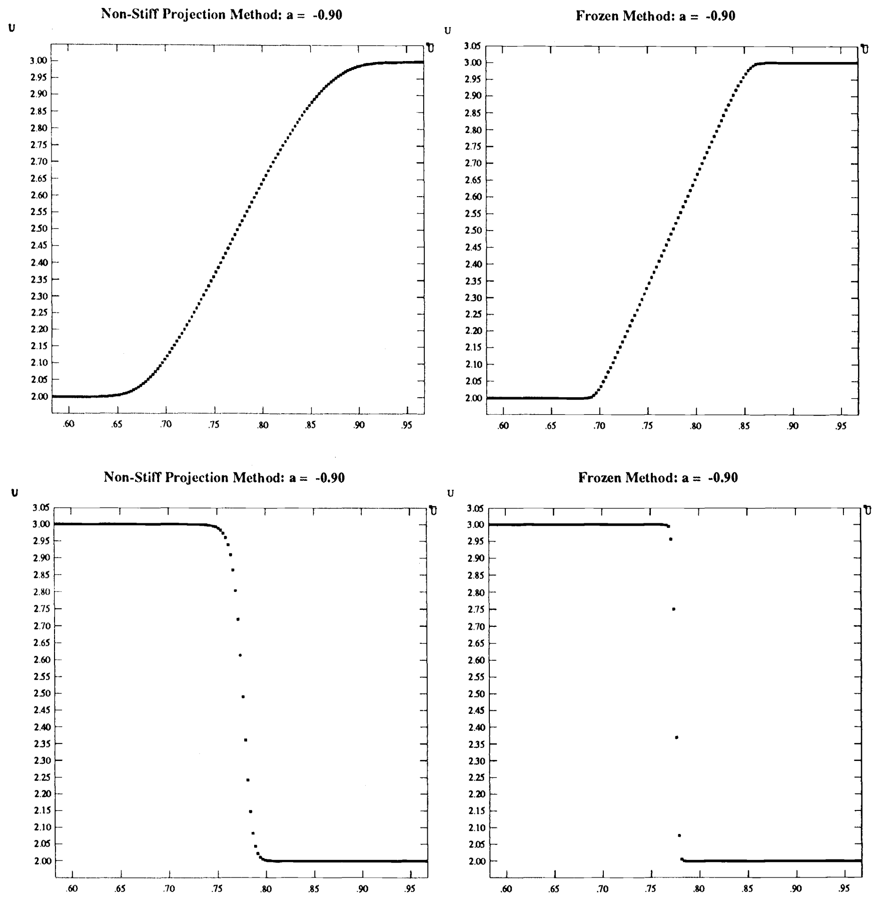


FIG. 2. *Numerical solutions of test shock and rarefaction problems by the frozen and the nonstiff projection methods for* $a = -.9$.

8. Numerical results: Eulerian gas dynamics with heat transfer. In this section we investigate using the method in §3 and the three variant methods in §4 to solve the Euler equations for gas dynamics, coupled with a simplified heat transfer rate equation, in one space dimension. The corresponding equilibrium equations are the Eulerian equations for isothermal flow. We formulate two initial value problems of the system whose time-asymptotic solutions are simple waves, an isothermal rarefaction, and an isothermal shock. We then look at three sets of numerical solutions of these two problems. The numerical results are consistent with the conclusions of the analysis in §6 for the model system.

8.1. Formulation of equations and test problems. We consider the Eulerian equations governing the one-dimensional flow of gas in contact with a constant temperature bath:

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uE+up\end{pmatrix}=\begin{pmatrix}0\\ 0\\ -K\rho(T-T_0)\end{pmatrix}. \tag{8.1}$$

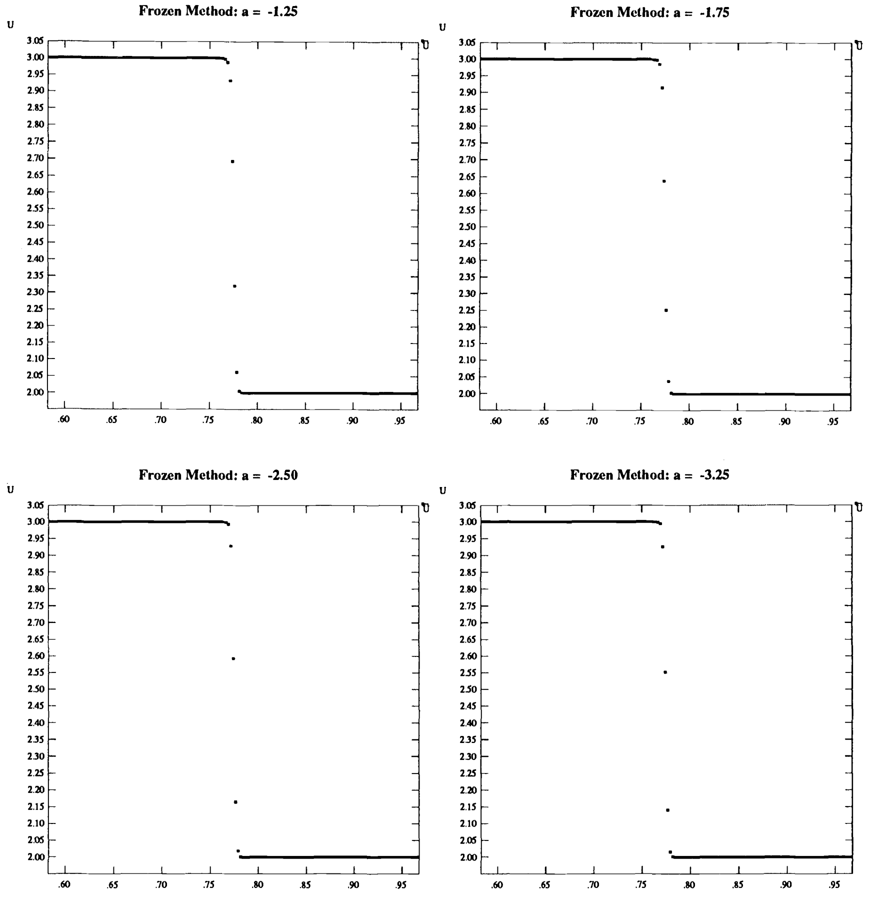


FIG. 3. *Numerical solutions of test shock problems by the frozen method for* $a = -1.25, -1.75, -2.5,$ *and* -3.25.

In this system, ρ is the density, u the velocity, $E = e + u^2/2$ the energy per unit mass, e the internal energy, T the temperature, and p the pressure. Away from equilibrium we assume that the gas is a γ-law gas, i.e., $p = (\gamma - 1)\rho e$. We choose units of temperature so that $T = e$. K and T_0 are positive constants. K is the heat transfer coefficient. T_0 is the temperature of the constant temperature bath. The frozen characteristic speeds of the system are $u - c$, u, and $u + c$, where $c = (\gamma p/\rho)^{1/2}$. At equilibrium, the flow is governed by the Eulerian equations for isothermal flow:

$$(8.2) \qquad \frac{\partial}{\partial t}\begin{pmatrix}\rho \\ \rho u\end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix}\rho u \\ \rho u^2 + p_*\end{pmatrix} = \begin{pmatrix}0 \\ 0\end{pmatrix}.$$

The pressure p_* is governed by an isothermal gas law: $p_*(\rho) = (\gamma - 1)\rho e_0$, where e_0 is the internal energy of the gas at $T = T_0$. The equilibrium characteristic speeds are $u - c_*$ and $u + c_*$, where

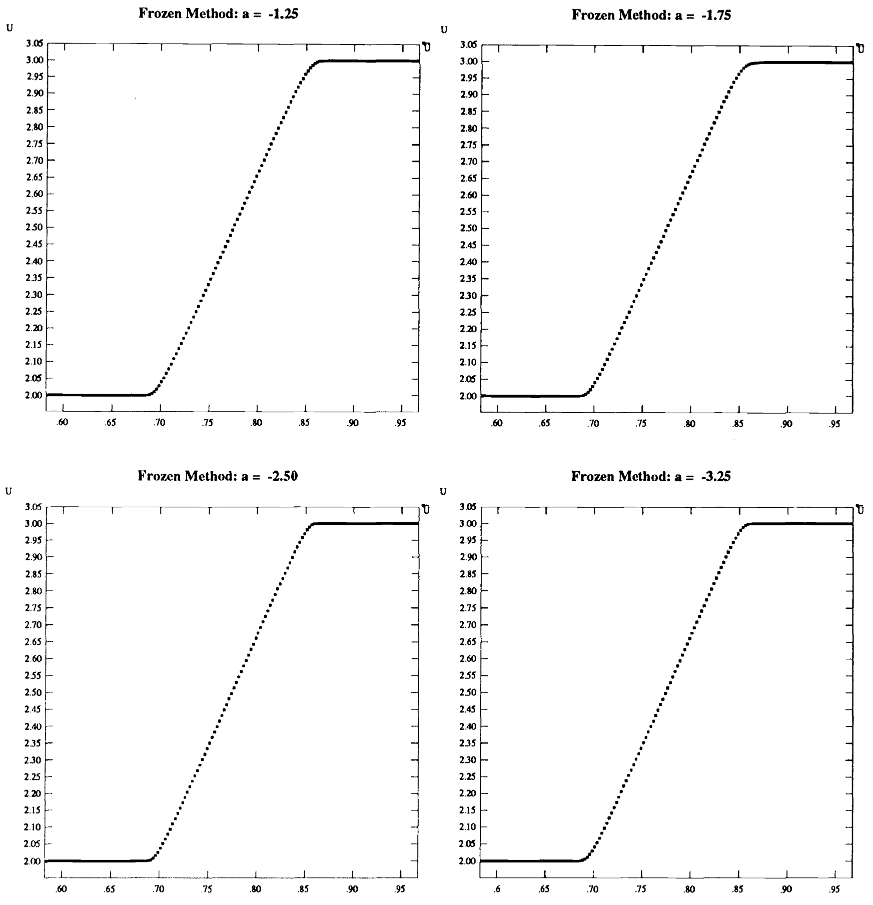


FIG. 4. *Numerical solutions of test rarefaction problems by the frozen method for* $a = -1.25, -1.75, -2.5,$ *and* -3.25.

$$c_* = \left(\frac{p}{\rho}\right)^{\frac{1}{2}} = \left((\gamma - 1)\, e_0\right)^{\frac{1}{2}}.$$

The frozen and the equilibrium characteristic speeds thus satisfy the subcharacteristic condition (1.1).

We now formulate two general initial value problems of (8.1). Under the assumption that Liu's conjecture [28] extends to (8.1), the solutions of these two problems tend to simple wave solutions of (8.2) as $K \to \infty$. We specify the solution at $t = 0$ in terms of the primitive variables ρ, u, and p as follows:

$$(8.3) \qquad \begin{aligned} (\rho,\ u,\ p)\,(x,0) &= (\rho_l,\ u_l,\ p_*(\rho_l))\,, \qquad x < x_0, \\ &= (\rho_r,\ u_r,\ p_*(\rho_l))\,, \qquad x > x_0. \end{aligned}$$

The left and right states of an initial value problem whose solution as $K \to \infty$ is an isothermal shock with speed s satisfy the following:

$$\begin{aligned} W &= ((\gamma-1)e_0\rho_l\rho_r)^{\frac{1}{2}}, \\ W(u_r-u_l) &= (\gamma-1)e_0(\rho_r-\rho_l), \\ p_l &= p_*(\rho_l), \\ p_r &= p_*(\rho_r), \\ s &= \frac{\rho_r u_r - \rho_l u_l}{\rho_r-\rho_l}. \end{aligned} \tag{8.4}$$

(W is the Lagrangian isothermal shock speed.) The left and the right states of an initial value problem whose solution as $K \to \infty$ is an isothermal rarefaction in the $u + c_*$ characteristic family satisfy these relations:

$$\begin{aligned} ((\gamma-1)e_0)^{\frac{1}{2}} \ln\tfrac{\rho_r}{\rho_l} &= u_r - u_l, \\ p_l &= p_*(\rho_l), \\ p_r &= p_*(\rho_r). \end{aligned} \tag{8.5}$$

We now use the results stated in the previous paragraph to formulate a set of test problems for the numerical methods. We define the constants in (8.1) as follows:

$$T_0 = e_0 = 1.0,$$

$$K = 10^8,$$

$$\gamma = 1.4.$$

At equilibrium, then, $c = .748$ and $c_* = .632$. Since $K >> 0$, i.e., the relaxation time is much smaller than the time scales associated with the fluid velocity and sound speed, we expect that a solution of (8.1) whose initial conditions satisfy (8.3) rapidly develops in time to a time-asymptotic solution of (8.1) which closely approximates a solution of the isothermal flow equations (8.2). The shock problems are defined as follows:

$$\begin{aligned} u_l &= u_r + .6, \\ \rho_l &= 2.5, \\ \rho_r &= 1.0, \\ p_l &= 1.0, \\ p_r &= .4. \end{aligned} \tag{8.6}$$

For a given value of u_r, we set the location x_0 of the initial discontinuity so that the location of the shock at $t = .4$ is $x = .81$. The rarefaction problems are defined as follows:

$$\begin{aligned} u_r &= u_l + .5795, \\ \rho_l &= 1.0, \\ \rho_r &= 2.5, \\ p_l &= .4, \\ p_r &= 1.0. \end{aligned} \tag{8.7}$$

For a given value of u_l, we set the location x_0 of the initial discontinuity so that the rarefaction wave extends from $x = .664$ to $x = .904$ at $t = .4$. We use a grid spacing of $\Delta x = .0025$ and a Courant number of $\sigma = .9$ in each run. We also use the following measure of error for the rarefaction problems so that we can compare the numerical results of the different methods:

$$\text{error} = \frac{1}{m} \sum_{j=1}^{m} \left| u_{\text{iso}}(x_j, t_n) - u_j^n \right| . \tag{8.8}$$

$u_{\text{iso}}(x_j, t_n)$ is the value of u, the fluid velocity, in the exact solution of the equations of isothermal flow (8.2) for the initial value problem given by (8.3) and (8.7), while u_j^n is the numerical approximation to the value of u in the solution at (x_j, t_n). m is the number of computational cells.

Remark. We need to make two statements about our claim that, given our particular choice of parameters, a solution of (8.1) whose initial conditions satisfy (8.3) rapidly tends to a time-asymptotic solution of (8.1) which closely approximates a solution of (8.2). First, there is an assumption that the spatial scale (and, by the CFL condition, the time scale as well) on which we want to resolve the solution is too small to fully resolve the relaxation process. We are assuming, then, not that K itself is large but that $K\Delta x$ is large compared to the magnitudes of the sound speeds and the fluid velocities, where Δx is the desired spatial resolution. Our second remark is that there are at this time no definitive theoretical results regarding the time-asymptotic behavior for $N \times N$ hyperbolic systems of conservation laws with relaxation where $N \geq 3$. There are definitive results only for 2×2 systems [8]. For our particular set of problems, however, the numerical results below do suggest that the time-asymptotic behavior of solutions of the frozen equations (8.1) is determined by the equilibrium system (8.2).

8.2. Frozen method. We use the frozen method of §3 to solve a series of isothermal rarefaction and shock problems of (8.1). The initial conditions of the rarefaction problems are given by (8.3), (8.7), and $u_l = -1.4, -.8, -.3, .3$, and $.8$. The initial conditions of the shock problems are given by (8.3), (8.7), and $u_r = -1.4, -.8, -.3, .3$, and $.8$. The resulting ten problems correspond to different relations among the frozen and the equilibrium characteristic speeds of the left and right states. For example, when $u_l = -1.4$, the frozen and the equilibrium characteristic speeds of the left and right states are all negative. The values of u in the numerical solutions of the rarefaction problems at $t = .4$ are displayed in the graphs in Fig. 5. The values of p in the numerical solutions of the shock problems at $t = .4$ are displayed in the graphs in Fig. 6. We can see from the figures that the results are consistent with the method being second-order accurate as $K \rightarrow \infty$, i.e., the frozen method produces sharply defined equilibrium shock and rarefaction solutions for all the test problems. Furthermore, the numerical results show approximately the same level of accuracy. The numerical solutions are oscillatory at discontinuities, however, in some of the shock problems for which the equilibrium and the

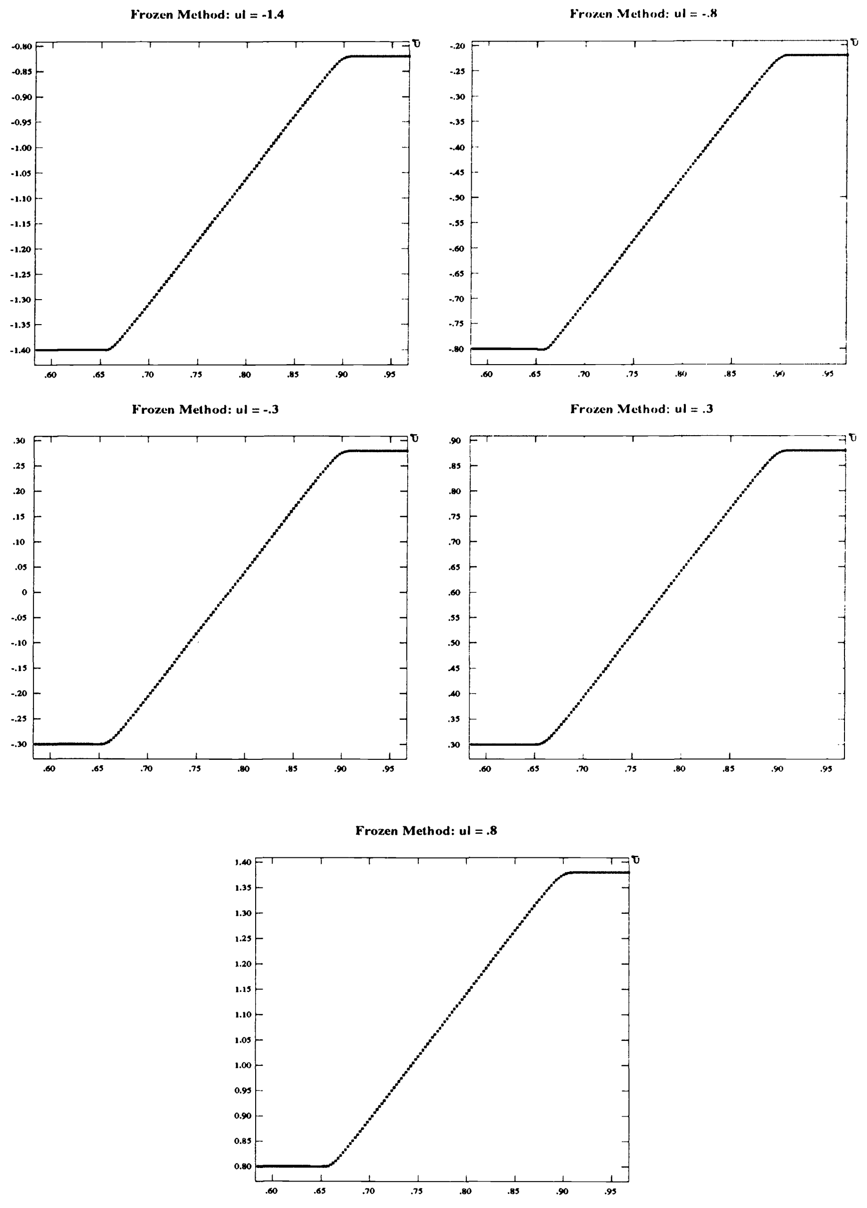


FIG. 5. *Numerical solutions of isothermal rarefaction problems by the frozen method for* $u_l = -1.4, -.8, -.3, .3,$ *and* $.8$.

frozen characteristic speeds do not all have the same sign, i.e., when $u_l = -.8$ and $-.3$. The errors at $t = .4$, as given by (8.8), in the solutions to the six rarefaction problems are, in order of increasing u_l, 2.185×10^{-3}, 2.261×10^{-3}, 2.534×10^{-3}, 2.506×10^{-3}, and 2.508×10^{-3}.

8.3. Fractional step and nonstiff projection methods. We use the fractional step and the nonstiff projection methods of §4 to solve two of the isothermal rarefaction problems discussed in §8.2. The initial conditions of the rarefaction problems are given by

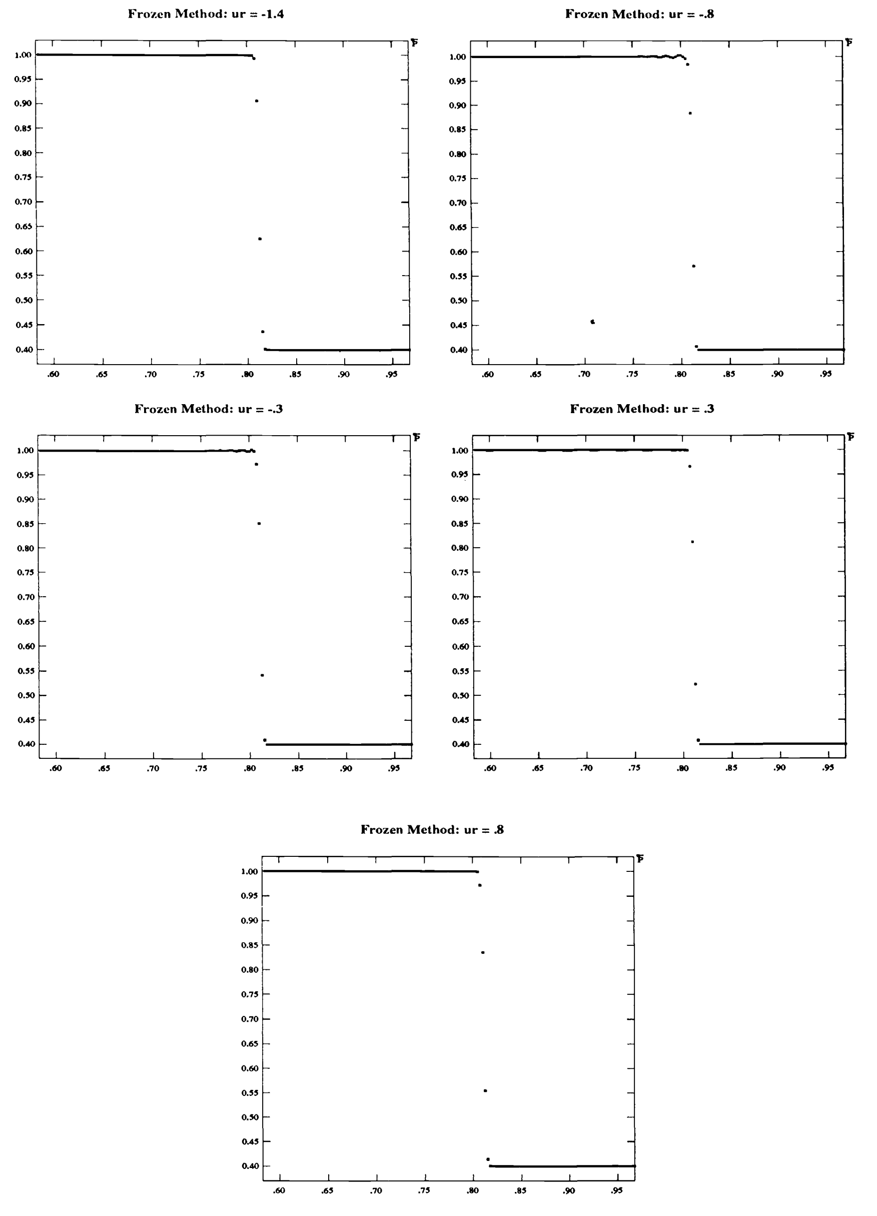


FIG. 6. *Numerical solutions of isothermal shock problems by the frozen method for* $u_r = -1.4, -.3, .3,$ *and* $.8$.

(8.3), (8.7), and $u_l = -.8$ and $-.3$. The values of u in the numerical solutions at $t = .4$ are displayed in the graphs in Fig. 7. We can see by comparing this figure with Fig. 5 that the results generated by the two methods are less accurate than those generated by the frozen method of §3. In particular, the positions of the trailing and leading edges of the rarefactions calculated by the fractional step method correspondingly trail and lead the same positions as calculated by the frozen method. The lower accuracy of the non-stiff projection method is even more apparent. We see that there are numerical "kinks" in the two solutions produced by this method. These appear where one of the frozen

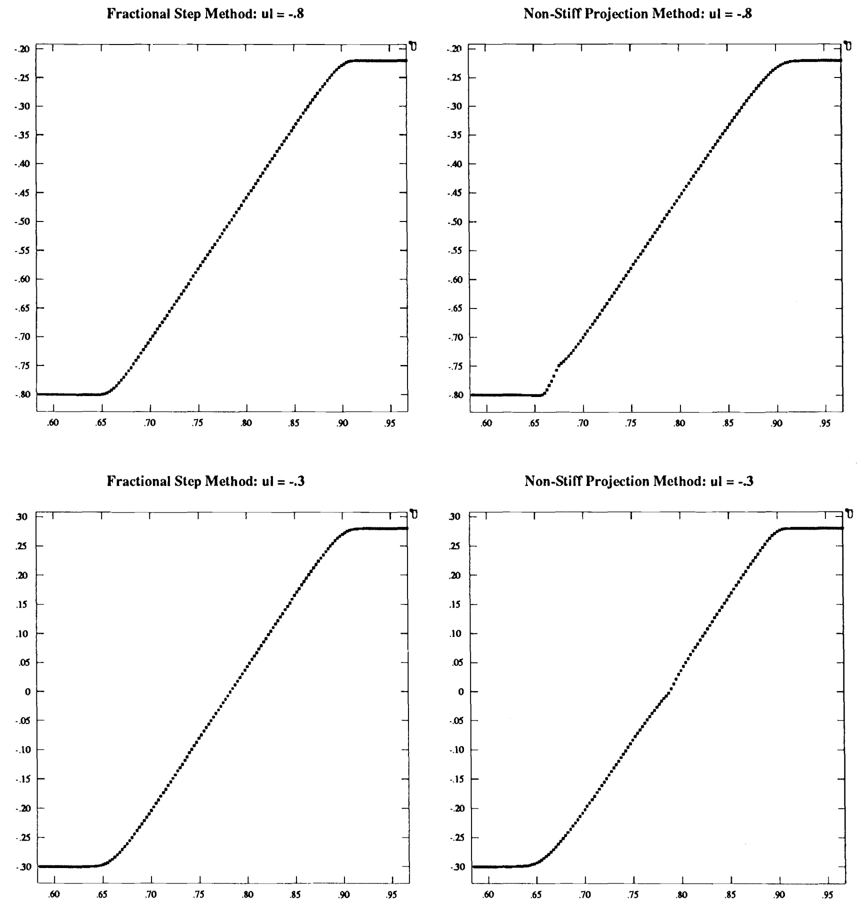


FIG. 7. *Numerical solutions of isothermal rarefaction problems by the fractional step and the nonstiff projection methods for* $u_l = -.8$ *and* $-.3$.

characteristic speeds u or $u + c$ equals 0. The errors at $t = .4$, as given by (8.8), in the two solutions produced by the fractional step method are 2.479×10^{-3} when $u_l = -.8$ and 2.867×10^{-3} when $u_l = -.3$. The corresponding values for the nonstiff projection method are 3.021×10^{-3} and 2.675×10^{-3}. The values for the frozen method, for comparison, are 2.261×10^{-3} and 2.534×10^{-3}.

8.4. Frozen method vs. equilibrium method. We use the equilibrium method of §4 to solve the two shock problems discussed in §8.2 for which the frozen method produced oscillations at the discontinuity. The initial conditions of the problems are given by (8.3), (8.6), and $u_r = -.8$ and $-.3$. The values of p in the numerical solutions at $t = .4$ are displayed in the graphs in Fig. 8; graphs of the solution produced by the frozen method are included for comparison. The solutions produced by the equilibrium method are less oscillatory than the solutions produced by the frozen method.

9. Additional considerations and extensions. In this section we discuss some considerations in developing a Riemann solver which satisfies (1.2). We also propose how to extend the methodology in §3 to the equations of gas flow with vibrational and

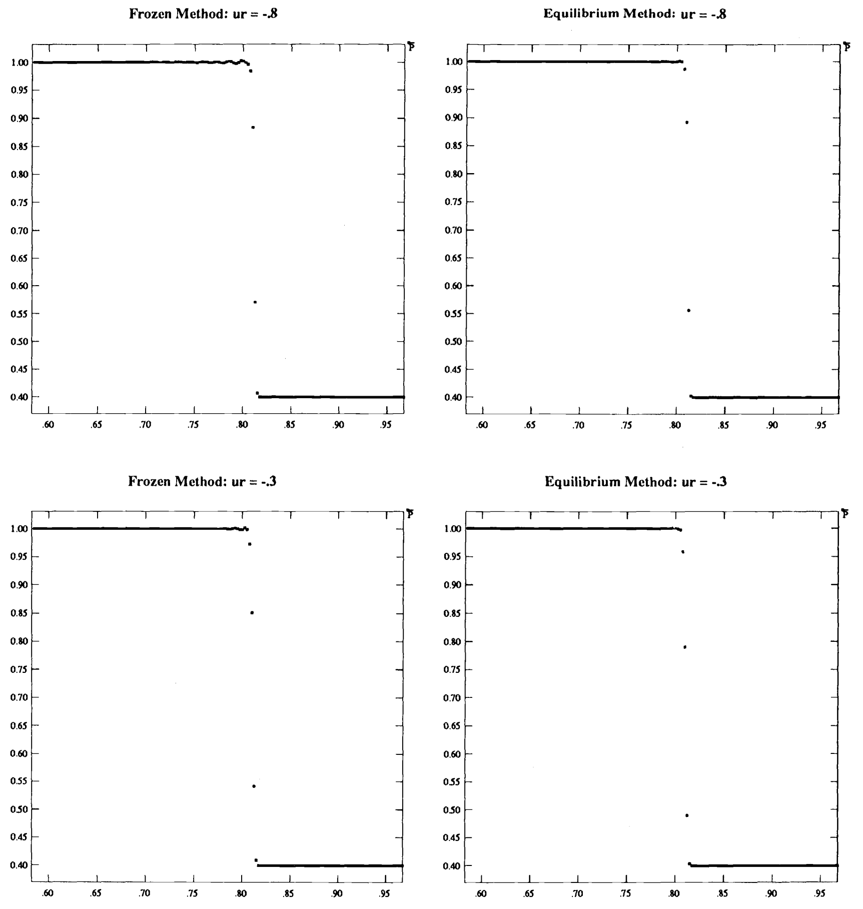


FIG. 8. *Numerical solutions of isothermal shock problems by the equilibrium and frozen methods for* $u_r = -.8$ *and* $-.3$.

chemical nonequilibrium, and, more generally, to systems in which one or more of the frozen characteristic speeds coincide.

9.1. Riemann solver. The numerical results in §8 demonstrate that a Riemann solver which satisfies (1.2) is a desirable component of a higher-order Godunov method for hyperbolic conservation laws with stiff relaxation. We now briefly evaluate five possible approaches to the development of such a Riemann solver.

One approach is simply to switch from a frozen Riemann solver to an equilibrium Riemann solver when $\tau/\Delta x$ drops below some threshold. We performed a numerical experiment to test this approach. We consider a traveling wave problem of (8.1) for which the states at $\pm\infty$ satisfy the isothermal shock relations (8.4). We numerically solve a series of traveling wave problems of this type in which K alone is varied. We use the frozen method of §3 and the equilibrium method of §4 to obtain solutions for $K = 50$, 400, and 10^8. The results are displayed in Fig. 9. For $K = 50$, the frozen method produces a reasonable numerical solution, while the equilibrium method does not. (Note that the exact solution at $K = 50$ is a traveling wave consisting of a shock followed by a smooth compression wave.) For $K = 10^8$, the equilibrium method produces a nonoscillatory

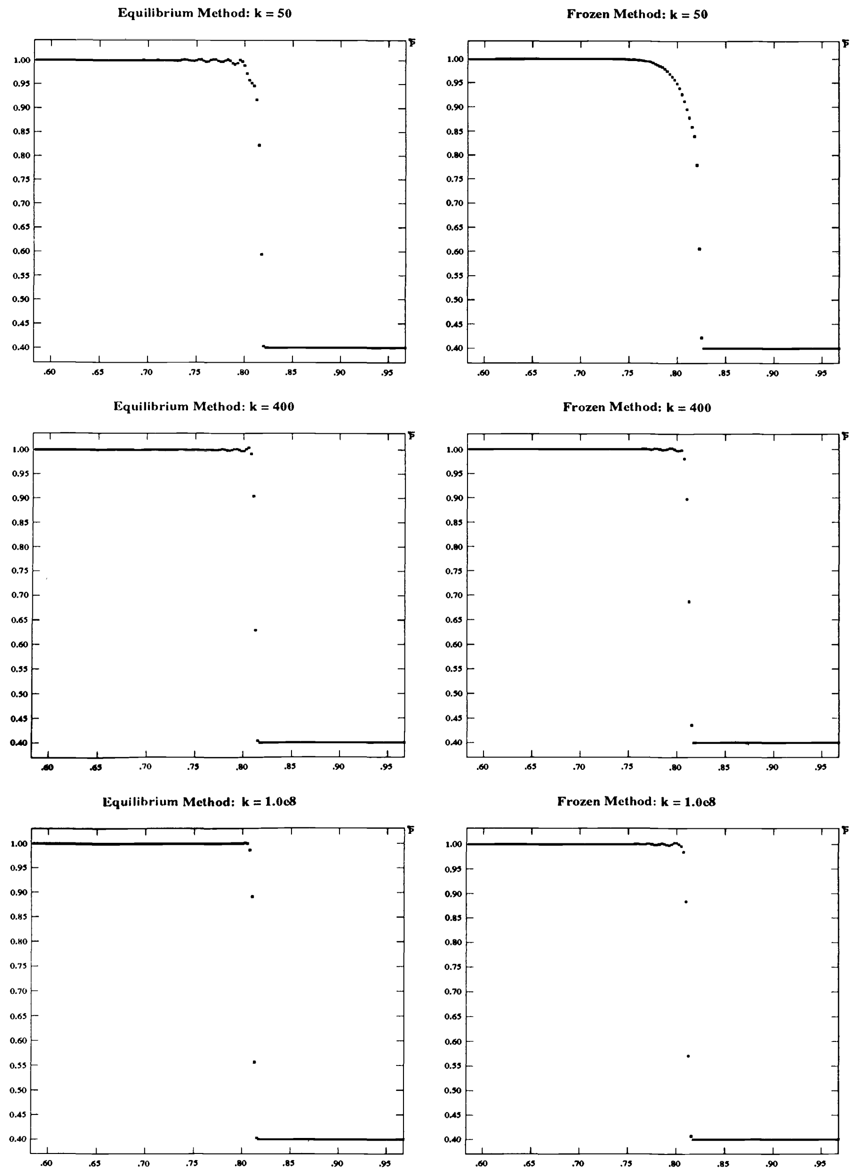


FIG. 9. *Numerical solutions of traveling wave problems by the equilibrium and the frozen methods for* $k = 50, 400, 10^8$.

representation of an isothermal shock, while the frozen method does not. However, for $K = 400$, both methods produce oscillatory results. Hence, it appears that a Riemann solver that simply switches between a frozen and an equilibrium Riemann solver depending on the size of the relaxation time would not significantly reduce the problem of numerical oscillations at shocks for nonzero, finite relaxation times. Furthermore, for a certain range of relaxation times, it appears that the Riemann solver must do something other than simply solve the frozen or the equilibrium Riemann problems.

The next three approaches also have shortcomings. One approach is to find the solution of the nonequilibrium system (3.1) for the initial value problem

$$U(x,0) = U^{n+\frac{1}{2}}_{j+\frac{1}{2},l}, \qquad x < 0, \tag{9.1}$$
$$= U^{n+\frac{1}{2}}_{j+\frac{1}{2},r}, \qquad x > 0$$

and evaluate the solution at $t = 0$. (Methods that use approaches similar to this are discussed in [3] and [17].) We do not actually gain anything by using this approach rather than the approach used in §3; however, because in effect its results are the same as a Riemann solver for the system with $S(U,x) \equiv H(U) \equiv 0$ except at equilibrium. Another approach is to find the time-asymptotic solution of the nonequilibrium system for the initial value problem (9.1) and evaluate that solution at $t = 0$. This approach has the opposite problem of always solving the Riemann problem for the equilibrium equation except when the relaxation time is infinite. A third approach is to somehow average the solutions of Riemann problems for the nonequilibrium equations and the equilibrium equations according to the value of the relaxation time. Although this approach works for zero and infinite relaxation times, it is unclear whether this technique is appropriate for intermediate values.

Another possible approach to this development could be based on redefining what is meant by an upwind method for a hyperbolic conservation law with relaxation. In particular, it may be the case that speeds other than the characteristic speeds should be used in determining upwind states. Linear [6], [22], [25], [39], [43] and nonlinear [8], [27] analyses suggest that this might be the case. In general, these analyses show that although wave fronts move with speeds determined by the frozen characteristic speeds, the overall signal carried by a wave moves at a speed intermediate to the frozen and equilibrium characteristic speeds. Perhaps one could apply these results in developing a numerical flux function which is appropriate for hyperbolic conservation laws with relaxation.

9.2. Flow with chemical and vibrational nonequilibrium. We assumed in developing the method in §3 that the frozen and equilibrium characteristic speeds were all distinct. This assumption is often not met by hyperbolic conservation laws with relaxation that are based on physical systems. In these systems, two or more of the frozen characteristic speeds often coincide. We now briefly examine one such system and show how to extend the method to that system.

We consider the equations describing gas flow with chemical and vibrational nonequilibrium. (See [39] for a more complete description.) For the sake of a simple exposition, we assume that the internal energy has two components, a translational component that is a function of temperature, e_{tr}, and a nonequilibrium component, q, i.e., $e = e_{\text{tr}} + q$. We assume q is governed by a rate equation which takes the following form in Lagrangian coordinates:

$$\frac{\partial q}{\partial t} = h(\rho, s, q) = \frac{q_*(\rho, s) - q}{\tau(\rho, s, q)}.$$

s, q_*, and τ are the entropy, the equilibrium value of q, and the relaxation time, respectively. The Eulerian equations of motion are then given by the following:

$$(9.2)\qquad \frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho E\\ \rho q\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uE+up\\ \rho uq\end{pmatrix}=\begin{pmatrix}0\\ 0\\ 0\\ \rho h(\rho,s,q)\end{pmatrix}.$$

The frozen characteristic speeds are $u-c$, u, u, and $u+c$, while the equilibrium speeds are $u-c_*$, u, and $u+c_*$. c and c_* are the frozen and the equilibrium speeds of sound. $c=(\partial p(\rho,s,q)/\partial\rho)^{1/2}$ and $c_*=(\partial p(\rho,s,q_*(\rho,s))/\partial\rho)^{1/2}$, where s is the entropy of the gas. We assume that $c_*<c$.

We now modify the method in §3 for the system of equations (9.2). Specifically, we modify the characteristic projection operators in the characteristic tracing step. We use the following criteria to determine which characteristic fields to include in the computation of $U^{n+1/2}_{j+1/2,l}$; the criteria for the computation of $U^{n+1/2}_{j+1/2,r}$ are similar:

(1) Use the $u-c$ characteristic field if $u_j>0$.

(2) Use the two u characteristic fields if $u_j+c_j>0$.

(3) Use the $u+c$ characteristic field if $u_j+c_j>0$.

Remark. The above methodology is easily extended to cases in which there are any number of nonequilibrium components of the internal energy, as long as the frozen and the equilibrium speeds of sound c and c_* satisfy $c_*<c$ for all possible states. However, if this inequality is not always satisfied (and, in particular, if any of the relaxation processes involved are exothermic in nature, such as in the case of chemically reacting flow), then it is necessary to resolve the time scales of the relaxation processes which contribute to the violation of the subcharacteristic condition. Otherwise, the numerical solution may contain nonphysical waves. See the remark at the end of §1.

10. Discussion and conclusions.

We have completed the initial stage of development of a second-order Godunov method for stiff hyperbolic systems of conservation laws with relaxation. Our method produces higher-order accurate solutions using time and space increments governed solely by the nonstiff part of the system, i.e., without fully resolving the effect of the stiff source terms. The method (§3) is a semi-implicit version of a second-order Godunov method [1], [11] for hyperbolic conservation laws with nonstiff source terms. Our method, which we refer to as the *frozen* method, differs from the nonstiff method in three ways. These differences are needed to ensure that the method is stable and second-order accurate for stiff problems.

(1) In the monotonized central difference calculation, two sets of expansion coefficients are calculated. One set is used in the calculation of left edge states in the characteristic tracing step; the other is used in finding the right edge states. The slope-limiting procedure is modified to account for the characteristic projection operator.

(2) In the characteristic tracing step, the stiff source term is treated in a semi-implicit fashion. Furthermore, the characteristic projection operators allow certain characteristic fields corresponding to negative frozen characteristic speeds to contribute to the value of left edge states, and certain fields corresponding to positive speeds to contribute to the value of right edge states.

(3) In the conservative differencing step, the stiff source term is treated in a semi-implicit fashion.

We note that these key components—the slope-limiting procedure, the characteristic projection operator, and the implicit differencing in the tracing and the conservative differencing steps—can be readily incorporated into the two-dimensional method described in [11].

The reasons for the modifications to the nonstiff method are easily stated. The stiff source term is treated semi-implicitly for stability. The modifications to the characteristic projection operators are necessary for the method to reduce to a second-order method for the equilibrium equation as the relaxation time tends to zero. As this limit is approached, disturbances carried by a given characteristic field of the equilibrium equation must be present in the two corresponding characteristic fields of the nonequilibrium system [27], [43]. Hence, the projection operator must take into account the signs of the equilibrium characteristic speeds as well as the signs of the frozen speeds. Given the modified projection operators, the modified slope-limiting procedure helps minimize oscillations at discontinuities in strongly nonlinear problems.

We have analyzed the method as applied to a simple model system and used the method to produce numerical solutions of the model system and of the Euler equations for gas dynamics with heat transfer. Our analysis shows that the method is second-order accurate for smooth solutions. Furthermore, analytical and numerical results show that the unsplit nature of the method and the modified characteristic projection operators are necessary for second-order accuracy as the relaxation time approaches zero.

Our method lacks one key desirable feature, namely, that it reduce to an upwind method for the equilibrium equation as the relaxation time tends to zero. We have shown that as $\tau \to 0$, the numerical results obtained with a method using a Riemann solver for the equilibrium equation are better than the results obtained with the frozen method in that the former are less oscillatory at shocks. These results suggest that the method in §3 would be improved if its Riemann solver satisfied the following: that it reduce to a Riemann solver for the equilibrium equation as $\tau \to 0$, and that the numerical method using the solver be an upwind scheme for all nonzero relaxation times. The development of such a Riemann solver is a topic for future work.

Acknowledgments. This work was done as part of a Ph.D. thesis under the guidance of Alexandre Chorin. I would like to thank him for his helpful suggestions, inspiration, and support. I would also like to thank the other two members of my dissertation committee, Phillip Colella and Ole Hald, as well as John Bell, Louis Howell, Gerry Puckett, and John Trangenstein, for helpful discussions during the course of my research. Finally, I would like to thank my wife, Camilla, whose love and support enabled me to finish this work.

REFERENCES

[1] J. B. Bell, P. Colella, and J. Trangenstein, *Higher order Godunov methods for general systems of hyperbolic conservation laws*, J. Comput. Phys., 82 (1989), pp. 362–397.

[2] J. B. Bell, P. Colella, J. A. Trangenstein, and M. Welcome, *Adaptive methods for high Mach number reacting flow*, AIAA Paper 87-1168-CP, in Proceedings, AIAA 8th Computational Fluid Dynamics Conference, Honolulu, HI, June 9–11, 1987, pp. 717–725.

[3] M. Ben-Artzi, *The generalized Riemann problem for reactive flows*, J. Comput. Phys., 81 (1989), pp. 70–101.

[4] J. H. Bolstad, J. Castor, P. Colella, P. Dykema, and R. Klein, *A semi-implicit Godunov method for coupling hydrodynamics to non-LTE radiative transfer*, in preparation.

[5] A. Bourlioux, A. Majda, and V. Roytburd, *Theoretical and numerical structures for unstable one-dimensional detonations*, SIAM J. Appl. Math., 51 (1991), pp. 303–343.

[6] L. Broer, *Characteristics of the equations of motion of a reacting gas*, J. Fluid Mech., 4 (1958), pp. 276–282.

[7] S. H. Chang, *On the application of subcell resolution to conservation laws with stiff source terms*, preprint.

[8] G. Chen, C. Levermore, and T. Liu, *Hyperbolic conservation laws with stiff relaxation terms and entropy*, Comm. Pure Appl. Math.,1992, submitted.

[9] A. CHORIN, *Random choice solutions of hyperbolic systems*, J. Comput. Phys., 22 (1976), pp. 517–533.

[10] ———, *Random choice methods with applications for reacting gas flows*, J. Comput. Phys., 25 (1977), pp. 253–272.

[11] P. COLELLA, *Multidimensional upwind methods for hyperbolic conservation laws*, J. Comput. Phys., 87 (1990), pp. 171–200.

[12] P. COLELLA AND H. GLAZ, *Efficient solution algorithms for the Riemann problem for real gases*, J. Comput. Phys., 59 (1985), pp. 264–289.

[13] P. COLELLA, A. MAJDA, AND V. ROYTBURD, *Theoretical and numerical structure for reacting shock waves*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1059–1080.

[14] P. COLELLA AND P. WOODWARD, *The piecewise parabolic method (PPM) for gas-dynamical simulations*, J. Comput. Phys., 54 (1984), pp. 174–201.

[15] E. ENGQUIST AND B. SJOGREEN, *Robust difference approximations of stiff inviscid detonation waves*, Tech. Rep. CAM Report 91-03, Dept. of Mathematics, Univ. of California, Los Angeles, 1991.

[16] H. GLAZ, P. COLELLA, J. COLLINS, AND R. FERGUSON, *High resolution calculations of unsteady, two-dimensional nonequilibrium gas dynamics with experimental comparisons*, AIAA Journal, 26 (1987), pp. 698–705.

[17] J. GLIMM, G. MARSHALL, AND B. PLOHR, *A generalized Riemann problem for quasi-one-dimensional gas flows*, Adv. in Appl. Math., 5 (1984), pp. 1–30.

[18] S. GODUNOV, *Difference methods for the numerical calculation of the equations of fluid dynamics*, Mat. Sb., 47 (1959), pp. 271–306. (In Russian.)

[19] E. HARABETIAN, *A subcell resolution method for viscous systems and computations of combustion waves*, J. Comput. Phys., 103 (1992), pp. 350–358.

[20] A. HARTEN, *Eno schemes with subcell resolution*, J. Comput. Phys., 83 (1989), pp. 148–184.

[21] A. HARTEN, P. LAX, AND B. VAN LEER, *On upstream differencing and Godunov-type schemes for hyperbolic conservation laws*, SIAM Rev., 25 (1981), pp. 35–61.

[22] W. HAYES AND R. PROBSTEIN, *Hypersonic Flow Theory*, Academic Press, New York, 1966.

[23] H. HOLLANDERS, L. MARRAFFA, J. MONTAGUE, P. MORICE, AND H. VIVIAND, *Computational methods for hypersonic flows: special techniques and real gas effects*, in Computation and Measurement of Hypersonic Flows, Birkhäuser, Basel, 1990.

[24] J. LAMBERT, *Computational Methods in Ordinary Differential Equations*, John Wiley and Sons, New York, 1973.

[25] L. LANDAU AND E. LIFSHITZ, *Fluid Mechanics*, Pergamon Press, Oxford, 1959.

[26] R. LEVEQUE AND H. YEE, *A study of numerical methods for hyperbolic conservation laws with stiff source terms*, J. Comput. Phys., 86 (1990), pp. 187–210.

[27] T. LIU, *Hyperbolic conservation laws with relaxation*, Comm. Math. Phys., 108 (1987), pp. 153–175.

[28] ———, *Geometric theory of shock waves*, in Multidimensional Hyperbolic Problems and Computations, J. Glimm and A. Majda, eds., Springer-Verlag, New York, 1990, pp. 198–202.

[29] A. MAJDA, *A qualitative model for dynamic combustion*, SIAM J. Appl. Math., 40 (1981), pp. 70–93.

[30] S. OSHER AND F. SOLOMON, *Upwind difference schemes for hyperbolic systems of conservation laws*, Math. Comput., 38 (1982), pp. 339–374.

[31] R. PEMBER, *Numerical methods for hyperbolic conservation laws with stiff relaxation*. I. *Spurious solutions*, UCRL JC-109097, Pt. I, Lawrence Livermore National Laboratory, Livermore, CA, December, 1991; SIAM J. Appl. Math., submitted, 1991.

[32] P. ROE, *Approximate Riemann solvers, parameter vectors, and difference schemes*, J. Comput. Phys., 43 (1981) pp. 357–372.

[33] G. STRANG, *On the construction and comparison of difference schemes*, SIAM J. Numer. Anal., 5 (1968), pp. 506–517.

[34] Z. TENG, A. CHORIN, AND R. LIU, *Riemann problems for reacting gas, with applications to transition*, SIAM J. Appl. Math., 42 (1982), pp. 964–981.

[35] B. VAN LEER, *Towards the ultimate conservative difference scheme*. III. *Upstream-centered finite difference schemes for ideal compressible flow*, J. Comput. Phys., 23 (1977), pp. 263–275.

[36] ———, *Towards the ultimate conservative difference scheme*. IV. *A new approach to numerical convection*, J. Comput. Phys., 23 (1977), pp. 276–299.

[37] ———, *Towards the ultimate conservative difference scheme*. V. *A second-order sequel to Godunov's method*, J. Comput. Phys., 32 (1979), pp. 101–136.

[38] ———, *On the relation between the upwind-differencing schemes of Godunov, Engquist–Osher, and Roe*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 1–20.

[39] W. VICENTI AND C. KRUGER, *Introduction to Physical Gas Dynamics*, John Wiley and Sons, New York, 1965.

[40] R. WARMING AND R. BEAM, *Upwind second-order difference schemes and applications in aerodynamic flows*, AIAA Journal, 14 (1976), pp. 1241–1249.

[41] P. WESSELING, *On the construction of accurate difference schemes for hyperbolic partial differential equations*, J. Engrg. Math., 7 (1973), pp. 19–31.

[42] G. B. WHITHAM, *Some comments on wave propagation and shock wave structure with application to magnetohydronamics*, Comm. Pure Appl. Math., XII (1959), pp. 113–158.

[43] ———, *Linear and Non-Linear Waves*, Wiley-Interscience, New York, 1974.

[44] P. WOODWARD AND P. COLELLA, *The numerical simulation of two-dimensional fluid flow with strong shocks*, J. Comput. Phys., 54 (1984), pp. 115–173.

[45] H. YEE, *A class of high-resolution explicit and implicit shock-capturing methods*, Tech. Rep., NASA Technical Memorandum 101088, 1989. Also published in a shorter version as "Upwind and Symmetric Shock-Capturing Schemes," NASA TM-89464, May, 1987.

[46] H. YEE AND J. SHINN, *Semi-implicit and fully implicit shock capturing methods for non-equilibrium flows*, AIAA Journal, 27 (1989), pp. 299–307.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 860–871, July 1993

OPTIMIZING THE NUMERICAL INTEGRATION OF INITIAL VALUE PROBLEMS IN SHOOTING METHODS FOR LINEAR BOUNDARY VALUE PROBLEMS*

L. RÁNDEZ†

Abstract. In this paper two new embedded Runge–Kutta (RK) methods RK5(4) specially designed for linear ODEs [L. F. Shampine, *J. Comput. Appl. Math.*, 15 (1986), pp. 293–300] are derived with (a) "small" principal truncation terms in the fifth-order formula, and (b) minimum number of Jacobian matrix evaluations per step. Numerical tests comparing their efficiency to the classical RKF5(4)#2 [E. Fehlberg, *Tech. Rep.* R-315, NASA, 1969], with the SUPORT code [M. R. Scott and H. A. Watts, *SIAM J. Numer. Anal.*, 1 (1977), pp. 40–70], are presented.

Key words. boundary value problems, shooting methods, Runge–Kutta methods

AMS subject classifications. 65L07, CR517

1. Introduction.

Let us consider the numerical solution of the linear boundary value problem (BVP)

$$(1.1)_a \qquad y'(t) = J(t)y(t) + g(t), \qquad t \in (a,b),$$

$$(1.1)_b \qquad B_a y(a) = \alpha, \qquad B_b y(b) = \beta,$$

where $y, g \in I\!R^m$, $J \in I\!R^{m\times m}$, $B_a \in I\!R^{(m-k)\times m}$, $B_b \in I\!R^{k\times m}$, $\alpha \in I\!R^{m-k}$, and $\beta \in I\!R^k$, by shooting methods. Such a class of methods reduces the BVP (1.1) to solving a linear system of equations involving the numerical integration of initial value problems. Usually, the method of superposition [1], [2], [5], [6], [9], [10], [13] is used to represent the solution as a combination of the fundamental and particular solutions, and it is necessary to integrate them numerically, e.g., by a Runge–Kutta (RK) method. The most popular method is the RK pair RKF4(5)#2 of formulas of orders 4 and 5 due to Fehlberg [4] and implemented as a BVP solver in the codes SUPORT [13] and MUSL and MUSN [1], [10].

In the numerical solution of (1.1) we can use algorithms for nonlinear problems, ignoring the linearity of the problem, but we cannot expect to obtain a particularly efficient algorithm in this way [6]. So, the most recent codes for solving BVP are split in linear and nonlinear problems (MUSL, MUSN, respectively [1]).

For nonlinear problems, the author [12] has proposed a combination of a RK triple for the differential equation and a special RK for the variational problem, but if it is used in linear problems, we can expect a loss of efficiency.

Since the initial value problem (IVP) to be integrated has the differential equation given by $(1.1)_a$, it is worth noting that a large portion of the cost of evaluating the derivative function comes from evaluating functions of the independent variable, i.e., mainly the Jacobian matrix $J(t)$. So, to advance from t_n to $t_n + h$, the RKF4(5)#2 requires evaluations at $t_n + c_i h$ where the c_i are $(0, \frac{1}{4}, \frac{3}{8}, \frac{12}{13}, 1, \frac{1}{2})$. Because no pair of successive evaluations is at the same point, this RK needs six Jacobian evaluations per step and the same number of stages.

The main idea of this paper is to exploit the linearity of the problem, because an important part of the multiple shooting method is the integration of the associated IVPs.

*Received by the editors November 13, 1991; accepted for publication (in revised form) July 28, 1992. This work was partially supported by Research Grant PS900121 DGICYT.

†Departamento de Matemática Aplicada, Universidad de Zaragoza, 50009 Zaragoza, Spain.

While Shampine [15] proposed using extrapolation with England's method, we prefer to develop new special explicit embedded pairs of RK formulas of orders 4 and 5 with six effective stages, but with a reduction in the number of Jacobian matrix evaluations involved in the process by using more than one function value corresponding to a single independent variable value, i.e., with several c_i's equal and consecutive. We construct two new pairs that require three and four evaluations of Jacobian matrix per step, respectively. Furthermore, the ratio between the size of the truncation error coefficients of the new pairs when compared with the RKF4(5)#2 are about 3 and 10 times smaller, respectively, for the new pairs.

These pairs are developed for general IVP, mainly because no substantial reduction is achieved in the number of order conditions for linear variable coefficient problems (the elementary differentials up to fourth order are the same, and only one for fifth order and three for sixth order are not present) and are also able to integrate nonlinear problems, for example, quasi linearization. A feature of the new formulas is the provision of a continuous solution, e.g. to plot the graph of the solution or to be used in a quasi linearization of a nonlinear problem.

The paper is organized as follows. In §2 we construct a family of RK pairs of orders 4 and 5 depending on four parameters such that it only requires four Jacobian evaluations per step. A similar analysis is carried out in §3, but now the RK pair requires only three Jacobian evaluations per step. In §4 a fourth-order continuous RK method is proposed for each method with no extra cost. Finally, in §4, some numerical tests showing the efficiency of the new pairs relative to Fehlberg's pair are presented using the code SUPORT [13].

2. The family of embedded RK5(4) methods. We are concerned with the numerical integration of the IVP associated with the BVP (1.1). In particular, we consider an IVP like

$$\left\{ \begin{array}{l} z'(t) = J(t)\, z(t) + g(t), \\ z(t_n) = z_n, \end{array} \right. \tag{2.1}$$

to be solved by an explicit RK method with six stages. The intermediate function evaluations are

$$g_i = J(t_n + c_i h) \left(z_n + h \sum_{j=1}^{i-1} a_{ij} g_j \right) + g(t_n + c_i h), \qquad i = 1, \ldots, 6, \tag{2.2}$$

and the approximation z_{n+1} to $z(t_{n+1})$ is given by

$$z_{n+1} = z_n + h \sum_{i=1}^{6} b_i g_i, \tag{2.3}$$

where h is the stepsize, and a_{ij}, b_i, and c_i are parameters defining the RK method. Using matrix notation $A = (a_{ij}) \in I\!R^{6\times 6}$, $b = (b_i)$, $c = (c_i) \in I\!R^6$, the procedure can be specified by its Butcher table of coefficients

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}. \tag{2.4}$$

It is important to observe from (2.2) that if the coefficients $c_i's$ are different, it is necessary to evaluate the Jacobian matrix $J(t)$ at least six times; but if $c_6 = 1$, the last evaluation of the current step can be re-used as the first one of the next step. So, the goal is the construction of RK methods with several coefficients $c_i's$ equal and consecutive in order to save Jacobian evaluations.

Throughout this paper, we will use the notation RK$p(q)l$ for an embedded pair of orders p and q $(p \geq q+1)$ requiring l Jacobian evaluations per step. Note that for the pair RKF5(4)#2, $p=5$, $q=4$, and $l=6$.

Let $z(t; t_n, z_n)$ be the local solution of (2.1) at the point (t_n, z_n). Then the local error of the method (2.2), (2.3) at the point t_{n+1} is defined by

$$e(t_{n+1}) = z(t_n + h; , t_n, z_n) - z_{n+1} = \sum_{\rho(\tau)\geq 1} C(\tau)F_n(\tau)\frac{h^{\rho(\tau)}}{\rho(\tau)!},$$

where τ denotes a rooted tree of order $\rho(\tau)$, the vector $F_n(\tau) \in \mathbb{R}^m$ is the elementary differential associated with τ at (t_n, z_n), and the scalar $C(\tau)$ is the corresponding weight. It must be remarked that $F_n(\tau)$ depends only on the differential equation (2.1) and the point (t_n, z_n), while $C(\tau)$ depends only on the parameters (A, b). The RK formula (2.2), (2.3) has order p if and only if $e(t_{n+1}) = O(h^{p+1})$.

To construct a family of RK methods of order 5 with six stages, assume that the parameters (A, b) satisfy

$$(2.5)_a \qquad b^TA = b^T - (b.c)^T,$$

$$(2.5)_b \qquad Ac = c^2/2 - (c_2^2/2)e_2,$$

$$(2.5)_c \qquad b_2 = 0,$$

where $c^j = (c_1^j, \dots, c_6^j)^T$, e_j is the vector with components $(e_j)_i = \delta_{ij}$, and $(u.v)$ denotes the componentwise product of vectors u and v, i.e., $(u.v)_i = u_iv_i$. Obviously, the last equation of $(2.5)_a$ implies $c_6 = 1$, so the last evaluation of the Jacobian matrix in the current step can be re-used in the following step. It can be verified that (2.2), (2.3) has order 5 if and only if $b_2 = 0$ and

$$(2.6) \qquad \begin{cases} b^Tc^j = 1/(j+1), & j = 0, \dots, 4, \\ b^T(Ac^2.c) = 1/15, \\ b^T(Ae_2.c) = 0. \end{cases}$$

Operating with the above set of equations, it is easy to show that there exists a family of methods with five free parameters $\overline{\gamma} = (c_2, c_3, c_4, c_5, a_{42})$.

In order to embed the fourth-order formula, the so-called FSAL (first same as last) condition is assumed. This means that for the fourth-order solution a new stage which is identical to the first evaluation of the next step is used. So, the table of coefficients for our pairs will have the form

$$\begin{array}{c|c} c & A \\ 1 & b^T \\ \hline & b^T \\ & \overline{b}^T \end{array} := \begin{array}{c|c} \overline{c} & \overline{A} \\ \hline & b^T \\ & \overline{b}^T \end{array},$$

with the solution of fourth order given by

$$(2.7) \qquad \overline{z}_{n+1} = z_n + h\sum_{i=1}^{7} \overline{b}_i g_i.$$

TABLE 1

0							
$\frac{31}{100}$	$\frac{31}{100}$						
$\frac{31}{100}$	$\frac{31}{200}$	$\frac{31}{200}$					
$\frac{38}{45}$	$-\frac{3838}{12555}$	$-\frac{1972504}{700569}$	$\frac{2778256}{700569}$				
$\frac{251}{272}$	$-\frac{1148730490751}{1647543185408}$	$-\frac{2040096907375}{336012097024}$	$\frac{158279273308125}{20202727333568}$	$-\frac{3656060398935}{25563492651008}$			
1	$-\frac{210141871}{259543036}$	$-\frac{60045500}{8013779}$	$\frac{16750217245000}{1784692624637}$	$\frac{283446135}{2198060332}$	$-\frac{28146694912}{132766728359}$		
1	$\frac{55581}{591356}$	0	$\frac{5986250000}{12861766359}$	$\frac{225534375}{245400428}$	$-\frac{47552180224}{63191167389}$	$\frac{8339}{30429}$	
order 5	$\frac{55581}{591356}$	0	$\frac{5986250000}{12861766359}$	$\frac{225534375}{245400428}$	$-\frac{47552180224}{63191167389}$	$\frac{8339}{30429}$	0
order 4	$\frac{344139203}{3839083152}$	0	$\frac{371432256250}{773135066691}$	$\frac{386648544825}{531046526192}$	$-\frac{1692035942272}{3798491284161}$	$\frac{21356179}{219494520}$	$\frac{1}{20}$

For the fourth-order solution (2.7) it is very easy to see that the order conditions reduce to $\bar{b}_2 = 0$ and

$$\left\{\begin{array}{ll} \bar{b}^T \bar{c}^j = 1/(j+1), & j = 0,\ldots,3, \\ \bar{b}^T \bar{A}\,\bar{c}^2 = 1/12, & \\ \bar{b}^T \bar{A}\, e_2 = 0. & \end{array}\right. \tag{2.8}$$

Clearly, the linear system (2.8) is satisfied by the fifth-order solution. So, we will use the value a_{42} to make (2.8) compatible. From the first five equations of (2.8), we have a linear system in $\bar{b}_1, \bar{b}_3, \ldots, \bar{b}_6$ whose matrix is singular if

$$c_4 = \frac{c_3}{2(5c_3^2 - 4c_3 + 1)}.$$

Solving the linear system and taking c_2, c_3, c_4, c_5, and $\bar{b}_7$ as parameters, we arrive at the value

$$a_{42} = \frac{c_4^2(3c_3 - 2c_4)}{2c_3c_2},$$

to compatibilize the coefficients of the fourth-order solution.

The choice of the parameters $\gamma = (c_2, c_3, c_4, c_5, \bar{b}_7)$ leads to the following cases studied in this paper.

(i) Taking the values $c_2 = c_3 = 31/100$, $c_4 = 38/45$, $c_5 = 251/272$, and $\bar{b}_7 = 1/20$ as tuning parameters of the embedded pair leads to the formula RK5(4)4 presented in Table 1. For this formula, the ℓ_2 norm of the principal truncation term in the fifth-order solution is $\|C_6(\tau,\gamma)\| = 3.20 \times 10^{-4}$.

(ii) Taking $c_2 = c_3 = (5-\sqrt{5})/10$, $c_4 = c_5 = (5+\sqrt{5})/10$, $b_5 = 1/5$, and $\bar{b}_5 = 11/50$ as tuning parameters of the embedded pair leads to the formula RK5(4)3 presented in Table 2. Here, we obtain $\|C_6(\tau,\gamma)\| = 9.92 \times 10^{-4}$.

TABLE 2

0							
$\frac{5-\sqrt{5}}{10}$	$\frac{5-\sqrt{5}}{10}$						
$\frac{5-\sqrt{5}}{10}$	$\frac{5-\sqrt{5}}{20}$	$\frac{5-\sqrt{5}}{20}$					
$\frac{5+\sqrt{5}}{10}$	$\frac{-\sqrt{5}}{10}$	$\frac{-2-\sqrt{5}}{2}$	$\frac{15+7\sqrt{5}}{10}$				
$\frac{5+\sqrt{5}}{10}$	$\frac{13\sqrt{5}}{120}$	$\frac{9+7\sqrt{5}}{16}$	$\frac{-70-41\sqrt{5}}{120}$	$\frac{25-5\sqrt{5}}{48}$			
1	$\frac{\sqrt{5}-1}{4}$	$\frac{\sqrt{5}}{2}$	$\frac{-\sqrt{5}}{2}$	$\frac{-\sqrt{5}+5}{100}$	$\frac{30-6\sqrt{5}}{25}$		
1	$\frac{1}{12}$	0	$\frac{5}{12}$	$\frac{13}{60}$	$\frac{1}{5}$	$\frac{1}{12}$	
order 5	$\frac{1}{12}$	0	$\frac{5}{12}$	$\frac{13}{60}$	$\frac{1}{5}$	$\frac{1}{12}$	0
order 4	$\frac{1}{12}$	0	$\frac{5}{12}$	$\frac{59}{300}$	$\frac{11}{50}$	$\frac{11-3\sqrt{5}}{240}$	$\frac{3+\sqrt{5}}{80}$

Note that for the formula RK5(4)3, the main drawback is that for quadrature problems, the local error estimate is zero. In this case, we can use the pair RK5(4)4 to avoid this deficiency, but in the numerical tests it has not been observed, since in general $J(t) \neq 0$.

On the other hand, we give in Table 3 some characteristic values of the new pairs proposed by Prince and Dormand [11] where

$$A^{(6)}(\gamma) = \|C_6(\tau,\gamma)\|,$$
$$B^{(6)}(\gamma) = \|\hat{C}_6(\tau,\gamma)\| \;/\; \|\hat{C}_5(\tau,\gamma)\|,$$
$$C^{(6)}(\gamma) = \|\hat{C}_6(\tau,\gamma) - C_6(\tau,\gamma)\| \;/\; \|\hat{C}_5(\tau,\gamma)\|,$$

and $\|C_{p+1}\|$ (or $\|\hat{C}_{p+1}\|$, $\|\hat{C}_{p+2}\|$) is the ℓ_2 norm of the coefficients in the expansion of the local error of z_{n+1} ($\bar{z}_{n+1}$, respectively), i.e.,

$$\|C_{p+1}(\tau,\gamma)\| = \sqrt{\sum_{\rho(\tau)=p+1} C(\tau,\gamma)^2}. \tag{2.9}$$

D is the largest coefficient of the RK method, S_R^p is the interval of absolute stability for the pth-order formula, and s is the number of stages.

TABLE 3

Method	$p(q)$	$A^{(6)}$	$B^{(6)}$	$C^{(6)}$	D	S_R^5	S_R^4	s
RKF5(4)#2	5(4)	0.00336	3.15	2.10	8.00	−3.6	−3.2	6
RK5(4)3	5(4)	0.00099	1.22	1.29	3.06	−3.1	−3.1	6(7)
RK5(4)4	5(4)	0.00032	1.65	1.69	9.38	−3.4	−3.4	6(7)

3. A fourth-order interpolant for the RK5(4) pairs. If the pairs developed in the last section are used in a quasi-linearization process, the provision of a continuous solution may be important. So, the aim of this section is to develop a fourth-order family of interpolants for the RK pairs of the above section with no additional function evaluations.

To construct such a continuous solution we follow the interpolatory approach of Shampine [14]. We have at our disposal

$$z_n, \;\; z_n' = J(t_n)z_n + g(t_n), \;\; z_{n+1}, \;\; z_{n+1}' = J(t_{n+1})z_{n+1} + g(t_{n+1}),$$

where z_{n+1}' and z_{n+1}' are $O(h^6)$ approximations to the local solution and its derivative at t_{n+1}, respectively. Since we are interested in a local Hermite interpolation polynomial of fourth degree in the interval $[t_n, t_n+h]$, we need an additional datum. We ask whether there are $\sigma \in (0,1)$ and $\hat{b}_i$, $i = 1,\ldots,7$ such that

$$z_{n+\sigma} = z_n + h\sum_{i=1}^{7} \hat{b}_i g_i \tag{3.1}$$

is a fourth-order approximation, i.e.,

$$\|z(t_n+\sigma h; t_n, z_n) - z_{n+\sigma}\| = O(h^5). \tag{3.2}$$

We have found that an approximation satisfying (3.1) and (3.2) exists for all $\sigma \in (0,1)$ with the coefficients $\hat{b}_i$ given as the solution of the linear system

$$\begin{pmatrix} c_3 & c_4 & c_5 & 1 \\ c_3^2 & c_4^2 & c_5^2 & 1 \\ c_3^3 & c_4^3 & c_5^3 & 1 \\ c_3^2 & c_4^2 & c_5^2 & 1 \\ a_{32} & a_{42} & a_{52} & a_{62} \end{pmatrix} \begin{pmatrix} \hat{b}_3 \\ \hat{b}_4 \\ \hat{b}_5 \\ \hat{b}_6 \end{pmatrix} = \begin{pmatrix} \sigma/2 - \hat{b}_7 \\ \sigma^2/3 - \hat{b}_7 \\ \sigma^3/4 - \hat{b}_7 \\ 0 \end{pmatrix},$$

in the case $c_2 = c_3$, where $\hat{b}_7$ is an arbitrary parameter, and

$$\begin{pmatrix} c_3 & c_4 & 1 & 1 \\ c_3^2 & c_4^2 & 1 & 1 \\ c_3^3 & c_4^3 & 1 & 1 \\ c_3^2 & c_4^2 & 1 & 1 \\ a_{32} & a_{42} & a_{62} & 0 \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} \sigma/2 \\ \sigma^2/3 \\ \sigma^3/4 \\ \hat{b}_5(a_{42} - a_{52}) \end{pmatrix},$$

in the case $c_2 = c_3$, $c_4 = c_5$, where $\hat{b}_5$ is an arbitrary parameter and the values of the vector $\hat{b}$ are given by

$$\hat{b}_3 = x_3, \quad \hat{b}_4 = x_4 - \hat{b}_5, \quad \hat{b}_6 = x_5, \quad \hat{b}_7 = x_6, \quad \hat{b}_1 = \sigma - \sum_{i=3}^{7} \hat{b}_i,$$

with $\sigma = \frac{1}{2}$.

We select the value of the free parameter to minimize the quantity

$$r^*(u) = \int_0^1 r(\theta)\, d\theta \quad \text{with} \quad r(\theta) = \frac{\|C_\theta(\tau)\|}{\|\hat{C}(\tau)\|},$$

where u is the free parameter.

Here, $\|\hat{C}(\tau)\| = 2.44 \times 10^{-3}$ is the ℓ_2 norm of the coefficients $\hat{C}$ of the elementary differentials of the local error of the fourth-order solution of the RK5(4)4 pair. For the RK5(4)3 pair we obtain $\|\hat{C}(\tau)\| = 2.85 \times 10^{-3}$, and $\|C_\theta(\tau)\|$ is the corresponding value of the continuous solution (3.1). These quantities have been computed with the help of the program developed in [8], which provides a FORTRAN code to evaluate the expressions (2.9) for a given RK formula for any order.

By a numerical search it was found that $r^*(\theta)$ is minimized for

$$\hat{b}_7 = 27/1000 \quad \text{for the RK5(4)4 and} \quad \hat{b}_5 = 13/2000 \quad \text{for the RK5(4)3},$$

and the coefficients $\hat{b}_i$ for each pair in $\sigma = \frac{1}{2}$ are shown in Table 4.

Finally, to show the quality of our interpolants we have plotted in Fig. 1 the functions $r(\theta)$ for the optimal interpolants. From Fig. 1, it is clear that the interpolant shows a very satisfactory behavior at intermediate points.

TABLE 4

$\hat{b}_i$	RK5(4)4	RK5(4)3
1	80930065829/383908315200	17/96
2	0	0
3	3647975007125/4638810400146	$(40+15\sqrt{5})/96$
4	−281400845553/2124186104768	$(4922-1875\sqrt{5})/12000$
5	63405740574112/284886846312075	13/2000
6	−2508796489/21949452000	$(-189\sqrt{5}+533)/9600$
7	27/1000	$(63\sqrt{5}-211)/3200$

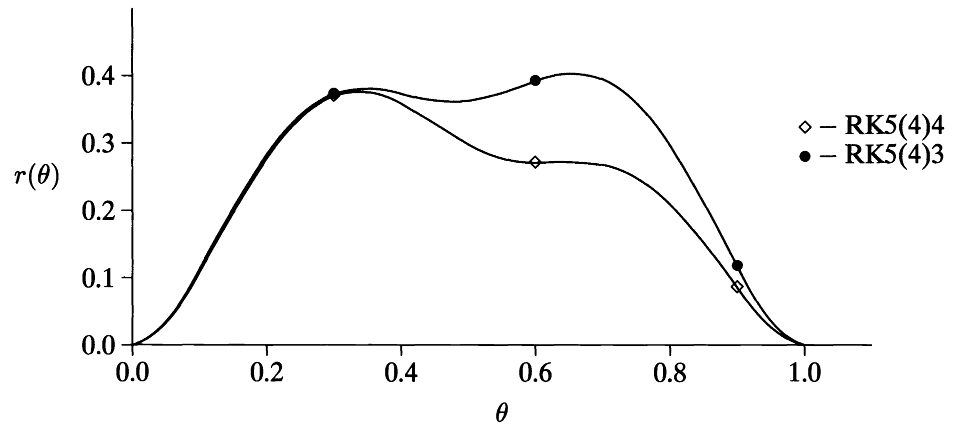


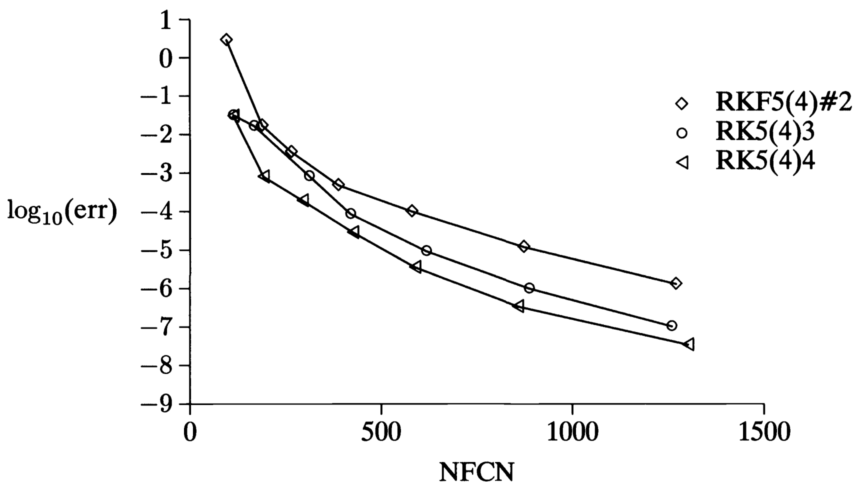
FIG. 1

4. Numerical experiments. The new RK5(4) pairs obtained in the above section have been tested on several IVPs, but we give only results obtained with the linear problem A3 from DETEST [3], which is

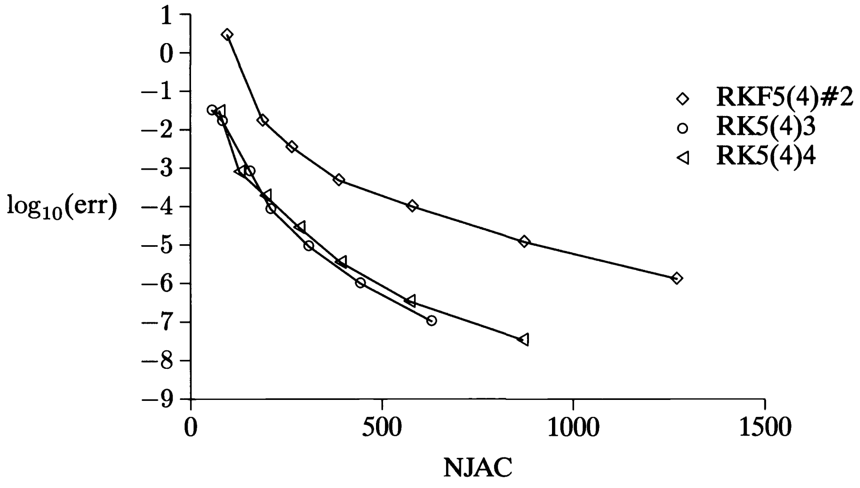
$$\begin{cases} y'(t) = \cos(t)y(t), & t \in (0,20), \\ y(0) = 1 \end{cases}$$

in unscaled form, for absolute tolerances in the range $10^{-2}, \dots, 10^{-8}$ and local extrapolation. Then, in Fig. 2(a) we plot, for each tolerance, the maximum global error $|y_n - y(t_n)|$ over the whole interval against NFCN, the number of function evaluations required. In Fig. 2(b) we plot the maximum global error against NJAC, the number of Jacobian evaluations.

Any code for solving linear BVPs, such as MUSL or SUPORT, is susceptible to a change of integrator, but as we have the SUPORT code of Scott and Watts [13], it was modified so that Fehlberg's pair and our new pairs might be used. With each pair, the numerical examples were integrated with a mixed absolute-relative control for tolerances $10^{-2}, \dots, 10^{-8}$. Briefly, we give results of two of the problems that appear in [13].



(a)



(b)

FIG. 2

Example 1. The first problem we wish to consider is

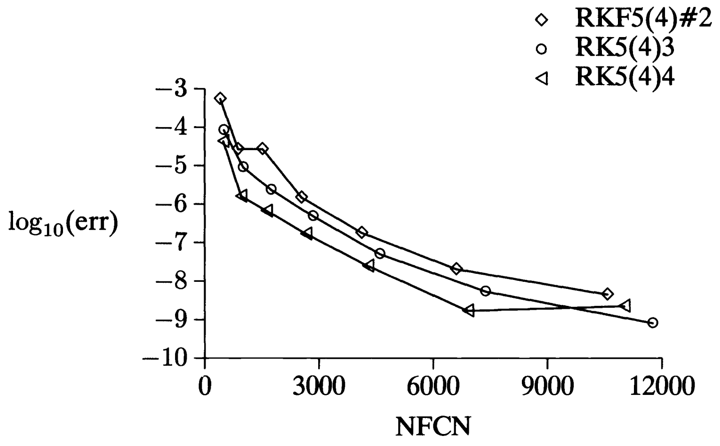
$$y''(t) + 3(\text{cotan}(x) + 2\tan(x))\, y'(t) + 0.7\, y(t) = 0,$$

subject to the boundary conditions

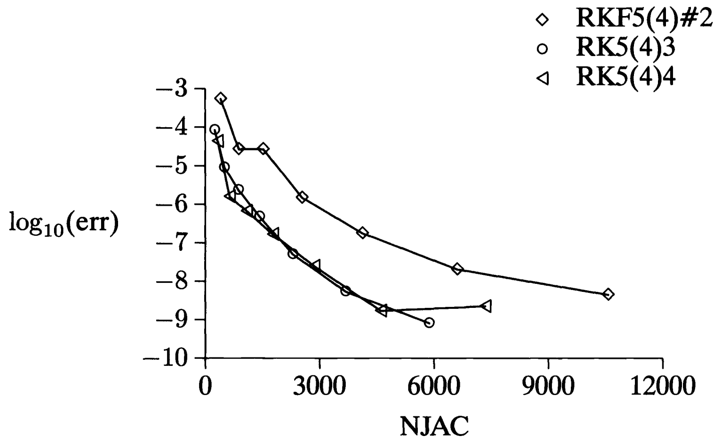
$$y(30^o) = 0, \qquad y(60^o) = 5.$$

It is worth noting that the independent variable t is taken in degrees, as in [13]. In this case, the problem is hard to solve due to a sharp spike in the solution. On the other hand, the two eigenvalues of the differential equation are negative. So, integrating the equations from 30° to 60°, the problem is easily solved. However, when the integration proceeds from 60° to 30° we have a difficult problem, due to the instability of the associated IVP. We give results only for the more difficult case.

In Fig. 3(a) we plot the maximum relative error in the solution at 15 points across the interval against NFCN, and in Fig. 3(b) we plot the maximum relative error against NJAC, the number of Jacobian evaluations. In all computations the number of orthonormalizations performed by the SUPORT code was 7.



(a)



(b)

FIG. 3

Example 2. Consider the boundary value problem [13]

$$y'' + \frac{3\epsilon}{(\epsilon + t^2)} y = 0,$$

$$y(-0.1) = -0.1(\epsilon + 0.01)^{-1/2},$$

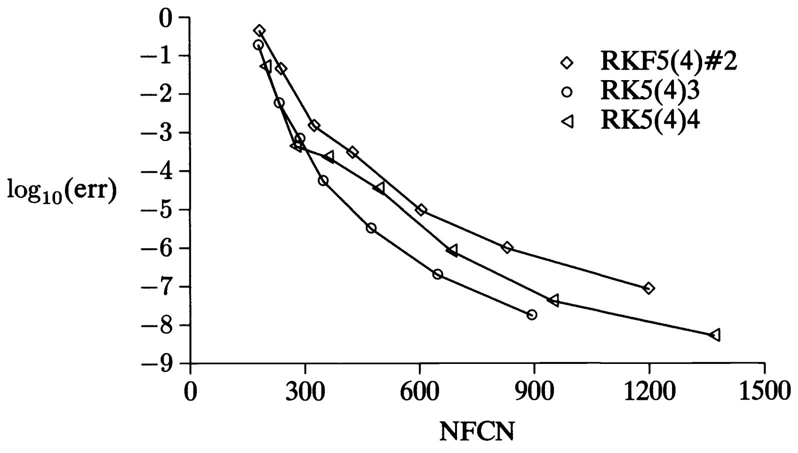
$$y(0.1) = -y(-0.1),$$

which has the exact solution

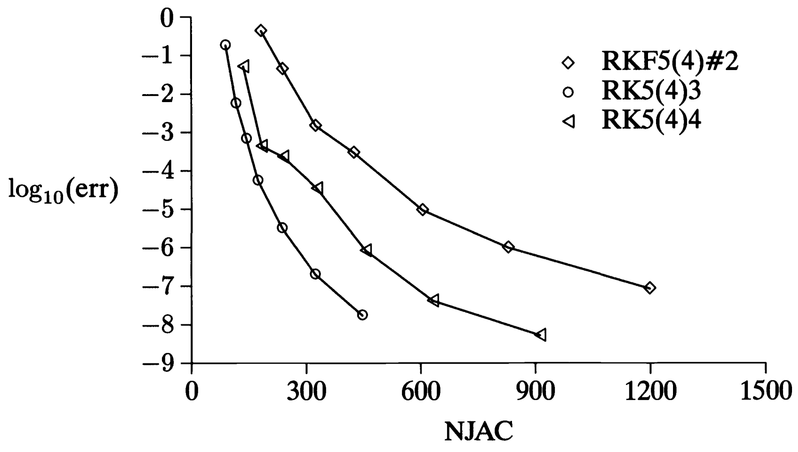
$$y(t) = t(\epsilon + t^2)^{-1/2}.$$

This problem was chosen by Scott and Watts [13] because it requires no orthonormalizations and it has a boundary layer of thickness $\sqrt{\epsilon}$ centered about $t = 0$. So, it is necessary for the code to have an efficient procedure for varying the stepsize of the integration.

The results of using the three integrators for $\epsilon = 10^{-6}$ are presented in Figs. 4(a) (global error against NFCN) and 4(b) (global error against NJAC). The maximum absolute error in the solution was sampled at 21 uniformly spaced points about the origin.



(a). $\epsilon = 10^{-6}$.



(b). $\epsilon = 10^{-6}$.

Fig. 4

The computations were carried out in double precision (16 significant digits) on a VAX STATION 3100 at the University of Zaragoza.

5. Conclusions. We feel that the solution of the initial value problem is a very important part of a shooting method code. So, with the new pairs developed in this paper, we get better numerical results with less computational cost. In the case of the RK5(4)3 pair, it uses 50% fewer Jacobian evaluations than the RKF5(4)#2 pair, while the size of the truncation error is about 3 times smaller; and the RK5(4)4 uses about 67% of the Jacobian evalutations used by RKF4(5)# 2, while the size of the truncation error is 10 times smaller.

In addition, the numerical experiments carried out in the preceding section show the importance of the study and construction of new special RK pairs for structured ODEs.

Finally, if intermediate values are required the two new pairs provide a fourth-order interpolant with no extra cost. In contrast, the RKF5(4)#2 requires one additional evaluation per step in order to obtain a fourth-order interpolant [7].

Acknowledgments. The author wishes to thank the referees and Prof. R. D. Skeel for their suggestions, which greatly improved the original paper.

REFERENCES

[1] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice Hall Series In Computational Mathematics, 1988.

[2] S. D. Conte, *The numerical solution of linear boundary value problems*, SIAM Rev., 8 (1966), pp. 309–321.

[3] W. H. Enright and J. D. Pryce, *Two FORTRAN packages for assessing initial value methods*, ACM Trans. Math. Software, 13 (1987), pp. 1–27.

[4] E. Fehlberg, *Low order classical Runge–Kutta formulas with stepsize control and their application to some heat transfer problems*, Tech. Rep. R–315, NASA, 1969.

[5] S. Godunov, *On the numerical solution of boundary-value problems for systems of linear ordinary differential equations*, Uspekhi Mat. Nauk., 16 (1961), pp. 171–174.

[6] G. Hall and J. M. Watt, *Modern Numerical Methods for Ordinary Differential Equations*, Clarendon Press, Oxford, 1976.

[7] M. K. Horn, *Fourth- and fifth-order, scaled Runge–Kutta algorithms for treating dense output*, SIAM J. Numer. Anal., 20 (1983), pp. 558–568.

[8] J. C. Jorge and L. Rández, *Un algoritmo para el cálculo de las condiciones de orden p, con p arbitrario para un método Runge–Kutta*, Publicaciones del Seminario Matemático García de Galdeano, Sección 1, N^0 11, 1989.

[9] H. B. Keller, *Numerical Solution of Two-Point BVP*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1976.

[10] R. M. M. Mattheij and G. W. M. Staarink, *An efficient algorithm for solving general linear two-point BVP*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 745–763.

[11] P. J. Prince and J. R. Dormand, *High order embedded Runge–Kutta formulae*, J. Comput. Appl. Math., 7 (1981), pp. 67–75.

[12] L. Rández, *Improving the efficiency of the multiple shooting technique*, Preprint Universidad de Zaragoza, Spain, 1991.

[13] M. R. Scott and H. A. Watts, *Computational solution of linear two-point BVP via orthonormalization*, SIAM J. Numer. Anal., 14 (1977), pp. 40–70.

[14] L. F. Shampine, *Interpolation for Runge–Kutta methods*, SIAM J. Numer. Anal., 22 (1985), pp. 1014–1027.

[15] ———, *Solution of structured non-stiff ODE's*, J. Comput. Appl. Math., 15 (1986), pp. 293–300.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 872–889, July 1993

ESTIMATING WAVEFORM RELAXATION CONVERGENCE*

B. LEIMKUHLER†

Abstract. Critical limitations of the waveform relaxation method concern the unsolved problems of estimating convergence and controlling the iteration. In this paper, waveform relaxation is applied to a linear, second-order model system and is studied in the Laplace domain. By placing the iteration in a weighted space, computable estimates for a window of rapid convergence are developed. Numerical experiments illustrate the utility of the approach.

Key words. waveform relaxation method, splittings, circuit simulation, parallel computing

AMS subject classification. 65L05

1. Introduction.

The waveform relaxation (WR) method can be applied to exploit parallel computers in the solution of large systems of ordinary differential equations (ODEs). The method was originally proposed for very large scale integration (VLSI) circuit simulation [8], but has since been adapted to problems in multibody dynamics [5] and to solving systems of spatially discretized partial differential equations [14].

WR relies on a splitting or decoupling of a given system of differential equations into a set of weakly coupled subsystems. At each *sweep*, the equations modelling each subsystem are solved independently in a fixed interval or *window*, treating the solutions or *waveforms* from other subsystems as input functions determined at the previous sweep or at an earlier stage of the current sweep. Although this paper is largely self-contained, the reader may find the book by Sangiovanni-Vincentelli and White [19] or papers by Miekkala and Nevanlinna [9], [10], useful background material.

The interest is in waveform relaxation as a parallel (or distributed processing) method for accelerating the time-domain simulation of a large ODE system. Although WR has not yet fulfilled its promise in terms of parallel speedups in general implementations, the method is highly competitive in many cases. The primary limiting factor in the development of waveform relaxation as a general method for VLSI circuit simulation has been the need for an efficient automatic partitioning algorithm that identifies a rapidly convergent splitting [16]. We do not present such an algorithm here, but we do examine conditions for the rapid and stable convergence of WR, which could be used as acceptance criteria in a partitioning algorithm.

We restrict ourselves primarily to splittings (decouplings) of the block Jacobi form which might be used in a parallel implementation. We avoid the issue of discretization here, relying on the work in [11] to insure that for sufficiently small stepsizes, the discrete version of the iteration will mimic the behavior of the continuous one.

In §2, a model linear second-order system is examined and several concepts (e.g., *stable convergence*) are discussed. Stable convergence of waveform relaxation is related to spectral properties of the iteration matrix in the Laplace transform of the waveform relaxation iteration. The matrices which define a linear system are decomposed using the *stamp representation* of electronic circuit simulation. Using the stamp representation, it is easy to obtain relations between the quadratic forms on the various matrices. A concept of the speed of the splitting is defined in terms of a proportionality constant between quadratic forms of the matrices defining the splitting.

*Received by the editors January 14, 1991; accepted for publication (in revised form) May 18, 1992.

†Department of Mathematics, University of Kansas, Lawrence, Kansas 66045. The work of this author was supported in part by the Academy of Finland.

The problem of determining the spectral radius of the Laplace domain iteration matrix $R(z)$ is then considered in §3. Sufficient conditions for $\rho(R(z)) \leq \omega$ are derived. These estimates generalize earlier estimates of [10] for convergence rate on an infinite interval.

Section 4 contains two numerical experiments. In particular, a curious relationship between block size and convergence rate for the equations describing a linear resistor-capacitor (RC) network (or, alternatively, the spatially discretized one-dimensional diffusion equation) is explained in terms of our theory, and a spring-mass particle system is used to illustrate the correctness of an estimate obtained for undamped problems.

2. Linear systems. In this section we describe the WR iteration, and outline the methods used for its analysis for a model class of linear ODE systems that includes multiparticle mechanical systems [3] and linear VLSI resistor-inductor-capacitor (RLC) circuits [8]. These results could also be extended to nonlinear problems via linearization or to include constraints (e.g., as indicated in [6] or [5]).

We consider here the following second-order problem:

$$M\ddot{p} = -Kp - D\dot{p} + f(t), \quad p(0) = p_0, \quad \dot{p}(0) = q_0. \tag{1}$$

For multiparticle systems, K and D are, respectively, spring and damper matrices which have $N \times N$ symmetric, positive semidefinite structure. For such a network, M, also $N \times N$, would typically be positive and diagonal, but we assume only positive definiteness for the moment. Note that the springs in (1) have zero rest states. In general, nonzero rest states lead to nonlinearities. It is also straightforward to interpret (1) in terms of RLC circuits, where the condition that M be positive definite can be interpreted to mean that there is a capacitor to ground from every node.

Given a splitting ($K = K^+ - K^-$, $D = D^+ - D^-$) of the system (1), we define the WR iteration process by the equation

$$\begin{aligned} &M\ddot{p}^{(k+1)} = -K^+p^{(k+1)} - D^+\dot{p}^{(k+1)} + K^-p^{(k)} + D^-\dot{p}^{(k)} + f(t), \\ &p^{(k+1)}(0) = p_0, \qquad \dot{p}^{(k+1)}(0) = q_0. \end{aligned} \tag{2}$$

Note that this framework does not allow the matrix M to be split.

Following the approach of [10] and [6], we analyze the iteration in the Laplace domain:

$$z^2M\hat{p}^{(k+1)} = -K^+\hat{p}^{(k+1)} - zD^+\hat{p}^{(k+1)} + K^-\hat{p}^{(k)} + zD^-\hat{p}^{(k)} + \phi,$$

where ϕ consists of terms involving $\hat{f}$ and the initial values, so that

$$\hat{p}^{(k+1)} = R(z)\hat{p}^{(k)} + \tilde{\phi}$$

with

$$R(z) = (z^2M + zD^+ + K^+)^{-1}(zD^- + K^-). \tag{3}$$

We will be concerned with spectral properties of $R(z)$.

2.1. Growth and the window of stable convergence. Although the iteration on finite intervals is always (eventually) superlinearly convergent, during the early sweeps of the iteration on long intervals, convergence may be quite slow or there may even be substantial growth; theoretically, all growth eventually settles down, but numerically speaking, the increase in the initial sweeps of the iteration can lead to serious instability. One problem is to estimate the length of a *window of stable convergence* wherein the growth during the early stages of the iteration is moderate in some sense. More generally, the problem is to estimate the length of a window wherein convergence is approximately geometric with a given rate of decay.

Although proper window length is a highly problem-dependent quantity, and it is apparent that any general-purpose program must incorporate an adaptive estimator for the proper window length, good theoretical estimates are needed to guide the development of such algorithmic devices.

Another motivation for pursuing this sort of spectral analysis of the WR iteration is for the purpose of making qualitative comparisons between splittings.

Separating nonphysical behavior in the computation of sweeps due to inaccuracies in the WR iteration from the true dynamics of the physical problem requires some estimation of the growth or decay mechanism of the iteration; here we use the exponentially weighted norm (following [13]) to try to find an interval where the convergence is rapid.

The starting point for our discussion is the formula of Proposition 2.1 (first shown in [10] in a slightly different setting) which affords us a means of computing the spectral radius of the iteration operator in a weighted space. Let $\mathcal{R}$ represent the iteration operator for (2).

PROPOSITION 2.1 (Miekkala–Nevanlinna). *Let $\rho_\xi(.)$ be the spectral radius in $L_{\xi,\infty}$, the ξ-weighted L_∞ space, $\xi > 0$, whose norm satisfies*

$$\|x\|_{\xi,\infty} = \sup_{t\in[0,\infty)} |e^{-\xi t}x(t)|,$$

with $|.|$ the Euclidean norm. Then

$$\rho_\xi(\mathcal{R}) = \sup_{\mathrm{Re}\{z\}>\xi} \rho(R(z)). \tag{4}$$

It can also be demonstrated that, in case the poles of $R(z)$ are located to the left of the vertical line $\mathrm{Re}(z) = \xi$, then the supremum in (4) is always achieved on that line.

We would like to choose a T for which $|\mathcal{R}^n f|_T$, the sup norm of the nth iterate on $[0, T]$, has moderate growth. It is convenient to treat f as a function whose domain is $[0, \infty)$ and whose support is in $[0, T]$. Then for $\xi \geq 0$,

$$\begin{aligned}|\mathcal{R}^n f|_T &= |e^{\xi t}e^{-\xi t}\mathcal{R}^n f|_T \\ &\leq e^{\xi T}\sup\nolimits_{0\leq t\leq T}|e^{-\xi t}\mathcal{R}^n f| \\ &\leq e^{\xi T}\sup\nolimits_{0\leq t\leq\infty}|e^{-\xi t}\mathcal{R}^n f| \\ &= e^{\xi T}\|\mathcal{R}^n f\|_{\xi,\infty} \\ &\leq e^{\xi T}\|\mathcal{R}^n\|_{\xi,\infty}\|f\|_{\xi,\infty} \\ &\leq e^{\xi T}\|\mathcal{R}^n\|_{\xi,\infty}|f|_T,\end{aligned}$$

by the assumption placed on f.

It is difficult to obtain simple, computable estimates for the norms of powers of $\mathcal{R}$, but, as we shall see in the next section, an estimate of the spectral radius is obtainable.

The sequence of norms of powers of $\mathcal{R}$ does not necessarily obey a uniform geometric decay. Although it is possible to obtain a bound on ξ which insures that $\|\mathcal{R}\|_{\xi,\infty} < \omega$ (see [17] and [7] for the techniques used in special cases), we expect spectral estimates to be somewhat finer, mimicing the situation for linear splittings of linear algebraic equations [20].

One can say that as $\operatorname{Re}\{z\} \to \infty$, the matrix $R(z)$ looks like an $O(1/z)$ perturbation to a normal matrix, which suggests that the spectral radius will, asymptotically as $T \to 0$, model the convergence rate of the iteration (even in the early sweeps).

DEFINITION 2.1. *The abscissa of ω-convergence is defined as*

$$\xi_\omega = \inf\{\xi : \rho_\xi(\mathcal{R}) < \omega\}.$$

Let $\xi = \xi_\omega$ and $T = T_\omega = \frac{1}{\xi_\omega}$. Then

$$|\mathcal{R}^n f|_{T_\omega} \tilde{<} e\omega^n |f|_{T_\omega},$$

where the symbol $\tilde{<}$ means "approximately bounded by." Up to the quality of this approximate bound, T_ω is an estimate for a window of convergence with rate ω.

If the concept of ω-convergence were formulated in the time domain, it would correspond to the notion of the "uniform convergence" defined by Sangiovanni–Vincentelli and White [19]. The translation of the results developed in the Laplace domain for the abscissa of ω-convergence to the time domain by simple inversion of the Laplace variable is a simplification, but as our numerical experiments show, enough of the important phenomena are captured by it for the technique to be of some usefulness for comparison of splittings and relative assessment of choices of window size.

2.2. Structure of linear systems: The stamp representation. We would like to restrict the class of splittings to a subset with practical benefit for parallel processing, namely, block Jacobi splittings. We first discuss the structure of the matrices involved in the description of (1) using *stamps*, and then describe the splitting matrices with the same tool.

We refer the reader to Collar and Simpson [3] for a discussion of formulating mechanical systems, and to Singal and Vlach [18] for a discussion of the stamp representation in the context of electronic circuits. The stamp representation is a powerful tool for decomposing the structure of a system of differential equations defined on a network.

We assume, first of all, that the network we are studying is connected. Furthermore, we assume that actions take place in some *relative coordinate frame*: one node of the network is distinguished and fixed. (The choice of this node is arbitrary, and there may be more than one fixed node.) Let N be the number of nonfixed nodes.

The flow of information in many network problems is determined by three types of elements: those (K) which relate solution values x, those (D) which relate the derivatives of the solution $\dot{x}$, and those (M) which relate second derivatives $\ddot{x}$. The equations describing the flow of information are obtained by summing the three types of contributions and setting the sum equal to any driving term f. The structures of the $N \times N$ matrices K and D and of M are precisely analogous, so we analyze only the first.

Let n_K elements of type K have positive coefficients $k_1, k_2, \ldots, k_{n_K}$. Suppose element i connects node l_i with node r_i. The matrix K may be written as a sum of simple rank-1 stamp matrices K_i, where, if neither terminal of element number i is fixed, the (l_i, l_i) and (r_i, r_i) elements of K_i are k_i, the (l_i, r_i) and (r_i, l_i) elements are $-k_i$, and the matrix is otherwise zero. (If either of the terminals of the element is a fixed node, this formula must be slightly modified: the associated stamp has just one nonzero element in the diagonal position corresponding to the index of the nonfixed terminal; its value is k_i.)

The block Jacobi splitting relaxes the interconnections between subsystems, so that D^+ and K^+ in (2) have a block diagonal structure (or may be permuted to a block diagonal matrix). We will assume that with a certain partitioning of the nodes of the network, the type M elements connect only nodes of a given block. We assume that all the interface elements (elements linking nodes of separated subsystems) connect a pair of nonfixed nodes.

We can very easily define the described splitting technique using stamps. If element i is a splitting element, let $\hat{K}_i$ represent the diagonal part of K_i. Let $Spl(K)$ denote the corresponding set of indices of the splitting elements. With i in $\{1, \dots, n_K\}$,

$$K = \sum_i K_i,$$

$$K^+ = \sum_{i \notin Spl(K)} K_i + \sum_{i \in Spl(K)} \hat{K}_i,$$

$$K^- = \sum_{i \in Spl(K)} (\hat{K}_i - K_i).$$

A similar decomposition may be used for D.

For future reference, we denote by $\mathcal{G}$ the graph that underlies the network, and by $\mathcal{G}_K$ ($\mathcal{G}_D$) the subgraph with nodes corresponding to all nodes of the network and edges corresponding to elements of type K (D). We always assume that the type-D and -K elements together describe a connected graph.

2.3. Bounds on quadratic forms. The equation satisfied by an eigenvector/eigenvalue pair (u, λ) of $R(z)$ is

$$(zD^- + K^-)u = \lambda(z^2M + zD^+ + K^+)u.$$

Multiplying by u^* and solving for λ, we have

$$\lambda = \frac{z\delta_- + \kappa_-}{z^2\mu + z\delta_+ + \kappa_+}, \tag{5}$$

where $\delta_\pm = u^*D^\pm u/(u^*u)$, $\kappa_\pm = u^*K^\pm u/(u^*u)$, and $\mu = u^*Mu/(u^*u)$. All these quantities are real, since they are quadratic forms on real, symmetric matrices. Note, too, that they are z-dependent, as the eigenvector u is dependent on z.

In the following propositions we give some facts about $\kappa_\pm$ and $\delta_\pm$.

PROPOSITION 2.2. *$\kappa_+ \geq |\kappa_-|$ and $\delta_+ \geq |\delta_-|$.*

Proof. The proofs of the two inequalities are precisely parallel; we consider the type-K case here.

Assume first for simplicity that none of the elements are connected to fixed nodes; we will see that this is of no consequence. Using the stamp representation of §2.2, we have

$$\begin{aligned}\kappa_+ &= \frac{1}{u^*u}\left[\sum_{i\notin Spl(K)} u^*K_iu + \sum_{i\in Spl(K)} u^*\hat{K}_iu\right]\\ &= \frac{1}{u^*u}\left[\sum_{i\notin Spl(K)} k_i(|u_{l_i}|^2 - 2\mathrm{Re}\{\bar{u}_{l_i}u_{r_i}\} + |u_{r_i}|^2) + \sum_{i\in Spl(K)} k_i(|u_{l_i}|^2 + |u_{r_i}|^2)\right]\\ &= \frac{1}{u^*u}(\eta + \sum_{i\in Spl(K)} k_i\sigma_i),\end{aligned}$$

where $\eta \geq 0$ and $\sigma_i \geq 0$. Similarly, we have

$$\begin{aligned}\kappa_- &= \frac{1}{u^*u}\sum_{i\in Spl(K)} u^*(\hat{K}_i - K_i)u\\ &= \frac{1}{u^*u}\sum_{i\in Spl(K)} 2k_i\mathrm{Re}\{\bar{u}_{l_i}u_{r_i}\}\\ &= \frac{1}{u^*u}\sum_{i\in Spl(K)} k_i\zeta_i.\end{aligned}$$

Now,

$$|\kappa_-| \leq \frac{1}{u^*u}\sum_{i\in Spl(K)} k_i|\zeta_i|.$$

By the Cauchy–Schwarz inequality, we may easily observe that $\eta \geq 0$ and that $|\zeta_i| \leq \sigma_i$, and this establishes the desired result.

Now if some of the (nonsplitting) elements were connections to fixed nodes, η in the above would be formulated slightly differently, but η would remain nonnegative, so the argument would go through unaltered. □

PROPOSITION 2.3. *κ_+ and δ_+ are never simultaneously zero.*

Proof. Assume the contrary, that $\kappa_+ = \delta_+ = 0$. By Proposition 2.2, $\kappa_- = \delta_- = 0$ and it follows that also

$$\frac{u^*Ku}{u^*u} = \kappa_+ - \kappa_- = 0, \qquad \frac{u^*Du}{u^*u} = \delta_+ - \delta_- = 0,$$

so that u is a singular vector of $K + D$.

The matrix $K + D$ is weakly diagonally dominant since it is the sum of stamp matrices. The associated graph of the matrix $K + D$ is precisely $\mathcal{G}$, the graph underlying the network; by assumption, this graph is connected. Furthermore, there is some fixed node. Let the ith node of the network be connected directly to this fixed node. From the stamp representation, it is evident that in the ith row of $K + D$, the diagonal element is strictly greater than the sums of the absolute values of the off-diagonal elements. Evidently the matrix $K + D$ is irreducibly diagonally dominant, and this contradicts the existence of a singular vector. □

Using these facts about the Rayleigh quotients, it is possible to locate the poles of $R(z)$.

PROPOSITION 2.4. *The poles of $R(z)$ all lie in the closed half-plane $\mathbf{C}_- = \{z : \text{Re}\{z\} \le 0\}$.*

Proof. If z is a pole of $R(z)$, then there is a $u \in \mathbf{C}^n$ such that $(Mz^2+D^+z+K^+)u = 0$. Taking Rayleigh quotients, letting $z = x + iy$ with $x, y \in \mathbf{R}$, and examining the real and imaginary parts, we obtain

$$\mu(x^2 - y^2) + x\delta_+ + \kappa_+ = 0, \qquad 2xy + y\delta_+ = 0.$$

Using the nonnegativity of the coefficients, we see that if $y \neq 0$, then the second equation implies that $x \le 0$, while if $y = 0$, then the first equation implies that $x \le 0$. $\square$

Note that in case there are only type-K elements present ($D = D^+ = D^- = 0$), poles occur at each of the points iy, where y is the square root of an eigenvalue of $M^{-1}K^+$. Observe that for (λ, u) an eigenpair of $M^{-1}K^+$, $M^{-1}K^+u = \lambda u$, so $M^{-1/2}K^+u = \lambda M^{1/2}u$, where $M^{-1/2}$ is real and symmetric positive definite (s.p.d.). Thus $M^{-1/2}K^+M^{-1/2}w = \lambda w$ where $w = M^{1/2}u$, implying that $\lambda \in \mathbf{R}_+$. The poles of $R(z)$ are continuous functions of the coefficients; as type-D elements are added to the system, it can be seen that the poles move continuously into the left half-plane.

2.4. Speed of a splitting. It is clear that ξ_ω will depend critically on the extent to which the system was naturally decoupled at the splitting; in other words, on the relative influence of the coupling elements. Our estimates for ξ_ω are determined based on the following concept of the *speed of the splitting.*

DEFINITION 2.2. *The speed of the splitting of the matrix K is the quantity*

$$\omega_K = \inf\left\{\omega \ge 0 : \inf_{u\in\mathbf{C}^n} u^*K^+u - \frac{1}{\omega}\left|u^*K^-u\right| \ge 0\right\}.$$

The speed of the splitting of the matrix D is defined analogously.

When $\omega_K = 0$, for example, then the subsystems are coupled only through type-D elements. From earlier propositions, it is apparent that ω_K and ω_D both lie in the intervals $[0, 1]$.

If K^+ is nonsingular, then we have simply

$$\omega_K = \rho((K^+)^{-1}K^-) = \rho((K^+)^{-1/2}K^-(K^+)^{-1/2}).$$

However, ω_K (or ω_D) may also be easily computed in singular cases if we use the graph structure to partition the matrices K^+ and K^-. Suppose K^+ and K^- are obtained as the splitting network in such a way that all elements of the splitting are connected between nodes of an index set I (the "interface" nodes). Let J be another set of indices constructed as follows: for each index $i \in I$, J contains i and the indices of all of the nodes that lie in the component of $\mathcal{G}_K$ to which node i belongs. We may assume without loss of generality that the indices in the set J are sequential beginning with 1. Then K^+ and K^- can be partitioned as follows:

$$K^+ = \begin{bmatrix} K_J^+ & 0 \\ 0 & K_2^+ \end{bmatrix}$$

and

$$K^- = \begin{bmatrix} K_J^- & 0 \\ 0 & 0 \end{bmatrix},$$

where $K_J^{\pm}$ corresponds to the indices in J. The matrix K_J^+ is weakly diagonally dominant; furthermore, in any connected component of the associated graph there is at least one node (a node of the interface) for which the corresponding diagonal element of K_J^+ is strictly greater than the sum of moduli of off-diagonal elements. This is sufficient to conclude that K_J^+ is positive definite (in particular, it is nonsingular). The speed of convergence is the least ω such that the matrix

$$\begin{bmatrix} K_J^+ - \frac{1}{\omega}K_J^- & 0 \\ 0 & K_2^+ \end{bmatrix}$$

is positive semidefinite. K_2^+ is always positive semidefinite. Evidently we must have $K_J^+ - \frac{1}{\omega}K_J^-$ positive semidefinite. The infimum over all such ω occurs when $\omega = \omega_K = \rho((K_J^+)^{-1}K_J^-)$.

3. Estimates for ξ_ω.

Let

$$r(z) = \frac{z\delta_- + \kappa_-}{\mu z^2 + z\delta_+ + \kappa_+},$$

where $z = x + iy$, with $x, y \in R$. In this section we treat the case $M = I$, so that $\mu = 1$. If $M \neq I$, we make the substitution $p = M^{-1/2}\bar{p}$ in (1), and premultiply by the equation by the same factor. This results in a simple row and column scaling of K and D. In this case, the estimate for ξ_ω given in Theorem 3.1, below, instead of depending on properties of $K^{\pm}$ and $D^{\pm}$, would depend on analogous properties of $M^{-1/2}K^{\pm}M^{-1/2}$ and $M^{-1/2}D^{\pm}M^{-1/2}$.

3.1. Upper bounds for ξ_ω.

For fixed ω, we derive computable conditions on $x \geq 0$ that insure that $\sup_{y\in\mathbf{R}} |r(x+iy)| \leq \omega$. Note that in light of Proposition 2.4, this condition insures that $\sup_{\mathrm{Re}\{z\}>x} \rho(R(z)) < \omega$ and hence that $x \geq \xi_\omega$.

Define $s(x,y) = |r(x+iy)|^2 = \frac{p(x,y)}{q(x,y)}$, where

$$p(x,y) = (x\delta_- + \kappa_-)^2 + (y\delta_-)^2$$

and

$$q(x,y) = (x^2 - y^2 + x\delta_+ + \kappa_+)^2 + (2x + \delta_+)^2 y^2.$$

If all of the coefficients were fixed, we could maximize $s(x,y)$ for y by solving the equation

$$\frac{\partial}{\partial y} s(x,y) = 0.$$

On the other hand, the coefficients are definitely *not* fixed with respect to x and y. Since we do not know the eigenvectors of $R(z)$, the best we can do is to seek a value of x which causes $s(x,y) \leq \omega^2$ when $y \in R$ and the coefficients of p and q are determined by *any* $u \in C^n$. The challenge, however, is that this estimate must be computable.

$s(x,y) \leq \omega^2$ when

$$q(x,y) - \frac{1}{\omega^2}p(x,y) \geq 0.$$

This leads to the relation

$$\begin{aligned}(6)\qquad &(x^2+y^2)^2 + \delta_+^2(x^2+y^2) + 2x\delta_+(x^2+y^2) + 2\kappa_+(x^2-y^2) + 2\delta_+\kappa_+x + \kappa_+^2 \\ &- \frac{1}{\omega^2}\left[\delta_-^2(x^2+y^2) + 2\delta_-\kappa_-x + \kappa_-^2\right] \geq 0.\end{aligned}$$

By appropriate regrouping of the terms in this expression, it is possible to develop sufficient conditions for $s(x,y) < \omega^2$. Assuming the following condition allows us to distinguish a case where the bound so derived appears to be particularly computable.

CONDITION 3.1. *When the quadratic forms are consistently defined (i.e., determined by the same vector $u \in \mathbf{C}^n$),*

$$\delta_+\kappa_+ - \frac{1}{\omega^2}\delta_-\kappa_- \geq 0.$$

Note that always

$$\delta_+\kappa_+ - \frac{1}{\omega_D\omega_K}\delta_-\kappa_- \geq 0,$$

so that if $\omega > \sqrt{\omega_D\omega_K}$, then Condition 3.1 holds, but this is a somewhat stronger assumption.

THEOREM 3.1. *Suppose that Condition* 3.1 *holds, then*

$$\xi_\omega \leq \max\{\theta_D\rho(D^-), \theta_K\sqrt{\rho(K^+)}, 0\},$$

where

$$\theta_D = \frac{1}{\omega} - \frac{1}{\omega_D}$$

and

$$\theta_K = \begin{cases} \left[2\left(3 - \sqrt{3^2 - \frac{\omega_K^2}{\omega^2}}\right)\right]^{\frac{1}{2}}, & 0 \leq \frac{\omega_K^2}{\omega^2} < 5, \\ \left[\frac{1}{2}\left(\frac{\omega_K^2}{\omega^2} - 1\right)\right]^{\frac{1}{2}}, & \frac{\omega_K^2}{\omega^2} \geq 5. \end{cases}$$

Proof. Regrouping the terms in (6), we obtain the relation

$$\begin{aligned}(x^2+y^2)\left((x+\delta_+)^2 - \frac{1}{\omega^2}\delta_-^2\right) + 2x\left(\delta_+\kappa_+ - \frac{1}{\omega^2}\delta_-\kappa_-\right)\\ + (x^2+y^2)y^2 + 2\kappa_+(x^2-y^2) + \kappa_+^2 - \frac{1}{\omega}^2\kappa_-^2 \geq 0.\end{aligned} \tag{7}$$

If $x \geq \theta_D\rho(D^-)$, then the first term is nonnegative. The second term is nonnegative when Condition 3.1 holds.

The third term is nonnegative when

$$\gamma(y^2;x) = y^4 + (x^2 - 2\kappa_+)y^2 + 2\kappa_+x^2 + \left(1 - \frac{\omega_K^2}{\omega^2}\right)\kappa_+^2 \geq 0.$$

Suppose first that $\omega_K^2/\omega^2 \leq 9$ and that

$$x^2 \geq 2\left(3 - \sqrt{3^2 - \frac{\omega_K^2}{\omega^2}}\right)\kappa_+;$$

then

$$\gamma(y^2;x) \geq y^4 + 2\left(2 - \sqrt{3^2 - \frac{\omega_K^2}{\omega^2}}\right)\kappa_+ y^2 + \left(13 - 4\sqrt{3^2 - \frac{\omega_K^2}{\omega^2}} - \frac{\omega_K^2}{\omega^2}\right)\kappa_+^2$$
$$= \left[y^2 + 2\left(2 - \sqrt{3^2 - \frac{\omega_K^2}{\omega^2}}\right)\kappa_+\right]^2 \geq 0.$$

On the other hand, if $\omega_K^2/\omega^2 \geq 1$, and if

$$x^2 \geq \frac{1}{2}\left(\frac{\omega_K^2}{\omega^2} - 1\right)\kappa_+,$$

then

$$\gamma(y^2;x) \geq y^4 + \left[\frac{1}{2}\left(\frac{\omega_K^2}{\omega^2} - 1\right) - 2\right]\kappa_+ y^2 + \left(\frac{\omega_K^2}{\omega^2} - 1\right)\kappa_+^2 + \left(1 - \frac{\omega_K^2}{\omega^2}\right)\kappa_+^2$$
$$= y^4 + \left[\frac{1}{2}\left(\frac{\omega_K^2}{\omega^2} - 1\right) - 2\right]\kappa_+ y^2.$$

This last expression is nonnegative regardless of y if $\omega_K^2/\omega^2 \leq 5$.

It is easy to see that

$$2\left(3 - \sqrt{3^2 - \frac{\omega_K^2}{\omega^2}}\right) \geq \frac{1}{2}\left(\frac{\omega_K^2}{\omega^2} - 1\right) \quad \text{if } \frac{\omega_K^2}{\omega^2} \in [5,9],$$

so the second choice is the better bound on ξ_ω whenever it is valid. □

Note that the final result only depends on the speeds of the splittings of K and D and on the spectral radii of K^+ and D^-. The result when $K = 0$ is of some special interest since it corresponds to the case of RC circuits or semidiscretized diffusion equations. More generally, we have the following useful special cases which follow directly from eliminating unnecessary steps in the proof of Theorem 3.1 for the cases $D^- = 0$ or $K^- = 0$.

THEOREM 3.2. *If* $K^- = 0$, *then*

$$\xi_\omega \leq \max\left\{(1/\omega - 1/\omega_D)\rho(D^-), 0\right\}.$$

If $D^- = 0$, *then*

$$\xi_\omega \leq \frac{1}{2}\frac{\sqrt{\omega_K}}{\omega}\sqrt{\rho(K^-)}.$$

Proof. In the first case $K^- = 0$, Condition 3.1 automatically holds and the third term on the left-hand side of (7) can be reorganized as

$$(y^4 - 2\kappa_+ y^2 + \kappa_+^2) + x^2(y^2 + 2\kappa_+).$$

Both of these terms are clearly nonnegative, regardless of x and y; all that is left is the first term in (7), which is nonnegative under the stated condition.

On the other hand, if $D^- = 0$, a sufficient condition for (7) to be satisfied can be written as

$$(x^2 + y^2)^2 + 2\kappa_+(x^2 - y^2) + \kappa_+^2 - \frac{1}{\omega^2}\kappa_-^2 \geq 0.$$

Reorder the terms in powers of y^2 to get

$$y^4 + 2(x^2 - \kappa_+)y^2 + (x^2 + \kappa_+)^2 - \frac{1}{\omega^2}\kappa_-^2 \geq 0.$$

Clearly, this is satisfied if

$$(x^2 + \kappa_+)^2 - \frac{1}{\omega^2}\kappa_-^2 \geq (x^2 - \kappa_+)^2.$$

Expanding the powers and cancelling, this reduces to

$$4\kappa_+ x^2 - \frac{1}{\omega^2}\kappa_-^2 \geq 0.$$

Noting that $\kappa_+ \geq \frac{1}{\omega_K}|\kappa_-|$, it is enough to demand

$$\frac{4}{\omega_K}|\kappa_-|x^2 - \frac{1}{\omega^2}\kappa_-^2 \geq 0,$$

which holds under the condition of the theorem. □

3.2. Estimates asymptotically correct as $\omega \to 0$. Although Theorems 3.1 and 3.2 provide insight into the WR process, our experience in computations suggests that a different line of reasoning can sometimes yield rather better estimates, particularly as $\omega \to 0$. To begin, let us examine the graph of $R(z)$ more closely. As a simple example, we consider the splitting of the spring network of Fig. 1 into "left" and "right" subsystems, each containing two of the mass points. With this splitting, $\rho(R(x+iy))$ is plotted in Fig. 2 against x and y. A second graphic in Fig. 3 shows the view along the x-axis looking towards the origin, demonstrating that the maximum over lines parallel to the imaginary axis ultimately occurs at $y = 0$ (for large enough x). This illustration is representative of the general case, and in fact we can say that if x is sufficiently large, the maximum value of $\rho(R(x+iy))$ over $y \in R$ occurs on the x-axis.

This means that if ω is sufficiently small, we may take $y = 0$, and this simplifies the calculation of ξ_ω.

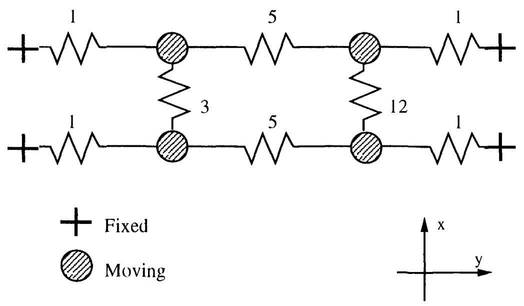


FIG. 1. *A simple spring-mass network.*

In what follows we assume $K^- \neq 0$. If $K^- = 0$, then one cannot easily improve on the estimate of Theorem 3.1.

The key idea is to rewrite the left-hand side of (6) as a polynomial in y^2:

$$\phi(y^2) = y^4 + by^2 + c,$$

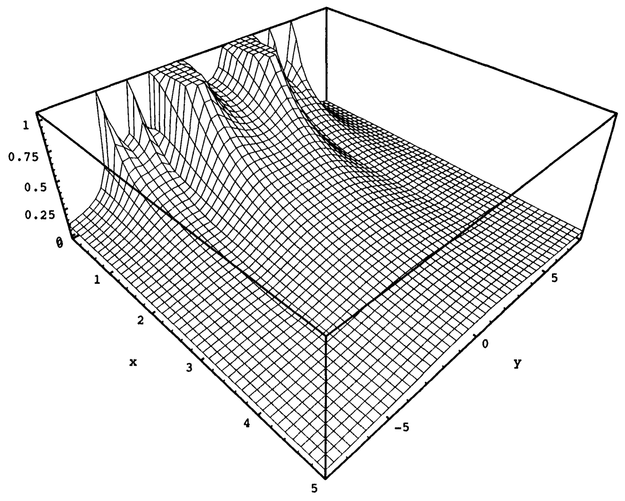


FIG. 2. *Plot of* $\rho(R(z))$.

where

$$b = b(x, u; \omega) = 2x^2 + \delta_+^2 + 2x\delta_+ - 2\kappa_+ - \frac{1}{\omega^2}\delta_-^2$$

and

$$c = c(x, u; \omega) = x^4 + \delta_+^2 x^2 + 2x^3\delta_+ + 2\kappa_+ x^2 + 2\delta_+\kappa_+ x + \kappa_+^2$$
$$-\frac{1}{\omega^2}\left(\delta_-^2 x^2 + 2\delta_-\kappa_- x + \kappa_-^2\right)$$

for $u \in C^n$, the vector on which the quadratic forms are consistently defined. $\phi(y) = 0$ for some y means that $\rho(R(x + iy)) \geq \omega$.

The following lemmas summarize the character of b and c.

LEMMA 3.3. *If* $\omega < \omega_K$ *there exists* x_ω *such that* (i) $c(x_\omega, u; \omega) \geq 0$ *for all* u, *and* (ii) $c(x_\omega, u; \omega) = 0$ *for some* u.

Proof. For x sufficiently large, the quartic term dominates and it is easy to see that $c(x, u; \omega) > 0$ for all u. On the other hand, $c(0, u; \omega) = \kappa_+^2 - \frac{1}{\omega^2}\kappa_- < 0$ for some choice of u, since $\omega < \omega_K$. $\min_u c(x, u; \omega)$ is defined and is a continuous function of x, and this guarantees the existence of an intermediate value x_ω with the required properties. □

LEMMA 3.4. *Let* $K^- \neq 0$. *If* $D^- \neq 0$, *then there exist positive constants* $\tilde{d}_0$ *and* $\bar{d}_0 \neq 0$ *such that* $\bar{d}_0 \geq \omega x_\omega \geq \tilde{d}_0$ *for* ω *sufficiently small.*

Proof. The proof of this fact is slightly technical, but the idea is simple. First, write $c(x_\omega, u; \omega) = c_+(x, u) + \frac{1}{\omega^2}c_-(x_\omega, u)$, where $c_-(x_\omega, u) = (\delta_- x + \kappa_-)^2$. If x_ω stays bounded as $\omega \to 0$, the only possibility is $\max_u c_-(x_\omega, u) \to 0$. Returning to the stamp representation for D^- and K^-, we know that these are both nonpositive matrices. Then taking u_0 to be a vector of all 1's, we see that $\max_u c_-(x_\omega, u) \geq (x_\omega d_0 + k_0)^2$ with nonnegative constants d_0 and k_0. Evidently, the only situation that would allow $\max_u c_-(x_\omega, u) \to 0$ is if $K^- = 0$, we have disallowed by an assumption. The only possibility is $x_\omega \to \infty$, as $\omega \to 0$.

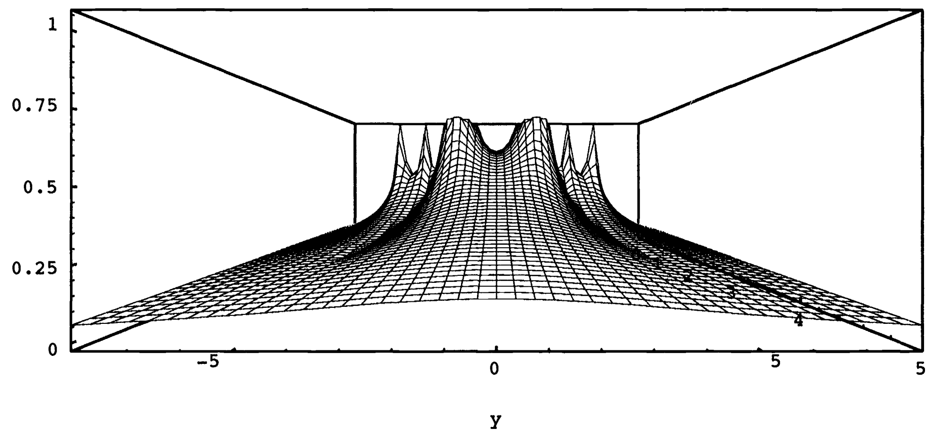


FIG. 3. *View toward O along the real axis.*

Now divide $c(x_\omega, u; \omega)$ by x_ω^4 to get

$$c(x_\omega, u; \omega) = 1 - \frac{\delta_-^2}{\omega^2 x_\omega^2} + O\left(\frac{1}{x_\omega}\right) + O\left(\frac{1}{\omega^2 x_\omega^3}\right).$$

If this is to be (i) nonnegative for all u and (ii) zero for some u, then the only possibility is that $\bar{d}_0 \geq \omega x_\omega \geq \tilde{d}_0$ for some nonnegative constants $\bar{d}_0$ and $\tilde{d}_0$. □

LEMMA 3.5. *For ω sufficiently small, $b(x_\omega, u; \omega) > 0$ for all u.*

The proof follows by grouping the terms in the expression for b as, e.g.,

$$b(x_\omega, u; \omega) = \left[\frac{1}{2}x_\omega^2 - 2\kappa_+\right] + \left[\frac{1}{2}x_\omega^2 + (x_\omega + \delta_+)^2 - \frac{1}{\omega^2}\delta_-^2\right].$$

The first term is ultimately positive since $x_\omega \to \infty$. The second term can be seen to be ultimately positive by use of the proof of the previous lemma.

LEMMA 3.6. *If ω is sufficiently small, then the maximum value of $\rho(R(x_\omega + iy))$ for all y is uniquely achieved at $y = 0$.*

Proof. For ω sufficiently small, $b(x_\omega, u; \omega) > 0$ independent of u. Since $c(x_\omega, u; \omega) \geq 0$, clearly $\phi(y) > 0$ for all $y \neq 0$. On the other hand, $\phi(0)$ reduces to the term $c(x_\omega, u; \omega)$, which we know is zero for some choice of u. In other words, $\rho(R(x_\omega)) \geq \omega$, and $\rho(R(x_\omega + iy)) < \omega$ for $y \neq 0$. Clearly, then, for ω sufficiently small, the maximum value of $\rho(R(x_\omega + iy))$ over y is achieved at $y = 0$. □

THEOREM 3.7. *Suppose $K^- \neq 0$. If $D^- \neq 0$, then, asymptotically as $\omega \to 0$,*

$$\xi_\omega \leq \max\{(1/\omega - 1/\omega_D)\rho(D^-), 0\}$$

(*independendent of K*).

If $D^- = 0$, then the asymptotic estimate becomes

$$\xi_\omega \leq \max\left\{\sqrt{(1/\omega - 1/\omega_K)\rho(K^-)}, 0\right\},$$

Proof. For ω sufficiently small, by Lemma 3.6, we may take $y = 0$ in (6), yielding

$$(8) \quad x^2\left((x + \delta_+)^2 - \frac{1}{\omega^2}\delta_-^2\right) + 2x\left(\delta_+\kappa_+ - \frac{1}{\omega^2}\delta_-\kappa_-\right) + 2\kappa_+ x^2 + \kappa_+^2 - \frac{1}{\omega^2}\kappa_-^2 \geq 0$$

as the condition for $x \geq \xi_\omega$.

If $D^- = 0$, then one requires only that

$$x^4 + 2\kappa_+ x^2 + \kappa_+^2 - \frac{1}{\omega^2}\kappa_-^2 \geq 0,$$

which reduces to the stated condition.

If, on the other hand, $D^- \neq 0$, divide the inequality (8) by x^2 to get

$$(9) \qquad (x+\delta_+)^2 - \frac{1}{\omega^2}\delta_-^2 + \frac{2}{x}\left(\delta_+\kappa_+ - \frac{1}{\omega^2}\delta_-\kappa_-\right) + 2\kappa_+ + \frac{\kappa_+^2 - \frac{1}{\omega^2}\kappa_-^2}{x^2} \geq 0.$$

Now fix $\epsilon > 0$. Define $\hat{x}_\omega = (1/\omega - 1/\omega_D)\rho(D^-)$ and let $x = (1+\epsilon)\hat{x}_\omega$, then the left-hand side of (9) becomes

$$(10) \qquad (\hat{x}_\omega + \delta_+)^2 - \frac{1}{\omega^2}\delta_-^2$$

$$+2\epsilon\hat{x}_\omega(\hat{x}_\omega + \delta_+) + \epsilon^2\hat{x}_\omega^2$$

$$+\frac{2}{x}\left(\delta_+\kappa_+ - \frac{1}{\omega^2}\delta_-\kappa_-\right)$$

$$(11) \qquad +2\kappa_+ + \frac{\kappa_+^2 - \frac{1}{\omega^2}\kappa_-^2}{x^2}.$$

Clearly the first term here is nonnegative. Furthermore, as $\omega \to 0$, making use of Lemma 3.4, the second term is positive and $O(\frac{1}{\omega^2})$, while the third and fourth terms are, respectively, $O(\frac{1}{\omega})$ and $O(1)$. Evidently, $x = (1+\epsilon)\hat{x}_\omega$ implies that $x \geq \xi_\omega$. Since ϵ was arbitrary, the best choice for the estimate is, asymptotically speaking, $\hat{x}_\omega$. $\square$

This confirms our observations in many computational experiments that the D-type elements dominate the convergence behavior of the iteration in the late sweeps or on a short window. On long windows, or in the early sweeps, it is our experience that the K-type elements are more important since they introduce oscillatory modes that are often difficult to resolve.

4. Numerical experiments.

4. Numerical experiments. We first detail some experiments with the simple RC interconnect (or RC line) of Fig. 4, and then turn to a multiparticle mechanical system.

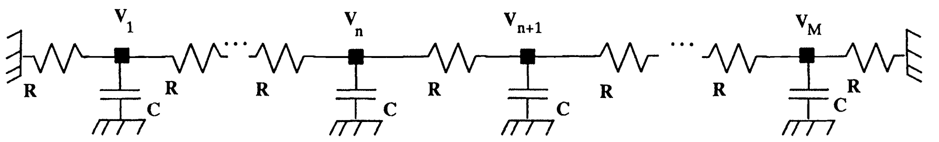


FIG. 4. RC *line.*

A problem similar to the RC line was considered in [10] and it has recently been examined in more detail by Leimkuhler and Ruehli [7] from another point of view. Here we compare the observed convergence rate of the iteration for several block sizes and several window sizes and attempt to relate our observations to the estimates obtained in the last section. In doing so, we must be careful to understand those estimates in

TABLE 1
Observed convergence rate for block G-J *iteration.*

Block size s	Time interval		
	[0,1]	[0,.5]	[0,.25]
1	.49/.38	.25/.20	.16/.12
2	.27/.22	.15/.12	.084/.064
4	.28/.21	.16/.12	.087/.065
30	.29/.21	.16/.12	.087/.065
60	.28/.21	.16/.12	.086/.065

the proper context: they are essentially heuristic in that they are based on a simplified transition from Laplace to time variable and on assumptions that are only strictly valid asymptotically as $\mathrm{Re}\{z\} \to \infty$ or as $\omega \to 0$. Despite the limitations, the examples of this section show that the estimates do have practical value.

The equations describing the nodal voltages in Fig. 4 are the familiar $\dot{x} = Ax$, where A is the tridiagonal matrix with 1, -2, and 1 on, respectively, the sub-, main, and super-diagonal. A pseudorandom initial vector was supplied to insure that all parts of the circuit were approximately equally active over the time interval. We used the backward Euler method and a sufficiently small fixed stepsize $h = 0.025$ so that the convergence results were not substantially different for smaller values of h (and hence mimic the behavior of the time-continuous iteration). In Table 1, we give our observations for the convergence rate of the iteration obtained by splitting the equations of the RC line of length 120 into various size blocks. The entries in the table are of the form r_3/r_6, where r_i represents the rate of convergence observed over the first i sweeps (computed as $(e_i/e_1)^{i-1}$, where e_i is the difference between the ith and $i-1$st iterates in maximum over both time and components). The pair of entries is just meant to give some indication of how much variation there is in the first few sweeps (bear in mind that the convergence rate will ultimately tend to 0 as $i \to \infty$, due to the superlinear convergence). We can also directly compute the spectral radius of the iteration matrix when $z = 0$, or ω_D for the purposes of our estimation technique, and establish that for the given problem, the spectral radius $\rho((D^+)^{-1}D^-)$ varies between about .983 and .997 in this case.

We were surprised to see that the rate of convergence fell so dramatically when the block size s changed from 1 to 2, while the change was nearly insignificant going from, e.g., $s = 2$ to $s = 4$, although this behavior has been observed in other computations [15]. We are prepared to give a partial explanation based on our estimates for convergence.

First note that there is substantial agreement between the values of spectral radius of $(D^+)^{-1}D^-$ for all cases. Thus no explanation is found in the existing spectral framework (in $L_2([0,\infty))$). On the other hand, computing $\rho(D^-)$ for each splitting, we find that $\rho(D^-) \approx 2$ for $s = 1$ and $\rho(D^-) = 1$, $s > 1$! This agrees qualitatively with the observations, since taking $C/T = \hat{\xi}_\omega = (1/\omega - 1/\omega_K)\rho(K^-)$, we have

$$\omega = \frac{T\rho(D^-)\omega_D}{T\rho(D^-) + C},$$

which is essentially proportional to $T\rho(D^-)$ in case (as here) $\omega_D \approx 1$ regardless of splitting and $T\rho(D_-)$ is small.

We can get some indication of how good of a *quantitative* window estimate $C/\hat{\xi}_\omega$ is by substituting an observed convergence rate r into $\hat{T} = C/\hat{\xi}_r$ and comparing with the time interval on which r was observed. These calculations have been performed in

TABLE 2
$C/\hat{\xi}_r$ for observed convergence rate r, $C = 2.7$.

Block size s	Time interval		
	[0,1]	[0,.5]	[0,.25]
1	1.31	.45	.26
2	1.00	.51	.25
4	1.05	.51	.26
30	1.10	.51	.26
60	1.05	.51	.26

Table 2; the number r_3 corresponding to the third sweep convergence rate estimate was used in each case; the number in the table should be compared to the endpoint of the time interval given at the top of the column in which it appears. We found that taking $C \approx 2.7$ gives reasonable agreement with the computations in all but the $s = 1, T = 1$ case. The results are certainly in keeping with our expectations, bearing in mind the points mentioned in the first paragraph of this section.

4.1. Multiparticle system. We conducted a large number of experiments with spring/mass networks. These problems were solved using an experimental multiparticle system solver developed by the author to serve as a software laboratory during the formulation of the mathematical theory. Here we summarize the results for Fig. 5 which were representative of our findings.

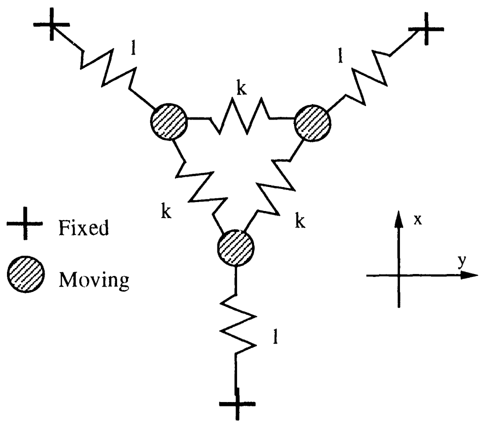


FIG. 5. *Spring-mass network.*

The network of Fig. 5 was split into three subsystems, each containing a single node. The splitting spring coefficient was varied from $k = 1$ to $k = 4$ and the integration was conducted on windows ranging from $[0, .3]$ to $[0, .9]$. The observed convergence rates (r_3) are summarized in Table 3.

We then inserted the estimates into the formula of Theorem 3.7 and scaled using the constant multiple $C = 3.9$. The results are given in Table 4. Again, there is reasonable agreement, given the substantial complexity of the phenomenon.

TABLE 3
Observed convergence rates for varying k.

Splitting spring k	Time interval						
	[0, .3]	[0,.4]	[0, .5]	[0,.6]	[0,.7]	[0,.8]	[0,.9]
1	.01	.03	.04	.05	.07	.08	.10
2	.02	.04	.06	.08	.10	.13	.16
3	.03	.06	.08	.11	.15	.19	.24
4	.04	.06	.10	.14	.19	.25	.32

TABLE 4
Computed estimates for convergence window.

Splitting spring k	Time interval						
	[0, .3]	[0,.4]	[0, .5]	[0,.6]	[0,.7]	[0,.8]	[0,.9]
1	.28	.49	.57	.64	.77	.83	.95
2	.28	.40	.50	.58	.66	.77	.87
3	.28	.40	.47	.57	.68	.79	.92
4	.28	.35	.46	.56	.68	.81	.98

5. Conclusion. The techniques used here can also be applied to handle constrained systems through a projection technique [5].

Although a priori estimates cannot tell the whole story, particularly as we turn to nonlinear systems and attempt to apply the spectral procedure to a linearization of the equations, the a priori estimates do give substantial useful information for little computational expense, and should guide the development of adaptive algorithmic estimates.

Acknowledgment. The author thanks Olavi Nevanlinna, Ulla Miekkala, Ralph Byers, and the referees for many thoughtful comments on the work.

REFERENCES

[1] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, North-Holland, Amsterdam, 1989.

[2] W. K. Chen, *Applied Graph Theory: Graphs and Electrical Networks*, 2nd ed., North-Holland Series in Applied Mathematics and Mechanics, North-Holland, Amsterdam, 1976.

[3] A. R. Collar and A. Simpson, *Matrices and Engineering Dynamics*, Ellis Horwood Limited, West Sussex, England, 1987.

[4] C. W. Gear, G. K. Gupta, and B. Leimkuhler, *Automatic integration of Euler–Lagrange equations with constraints*, J. Comput. Appl. Math., 12/13 (1985), pp. 77–90.

[5] B. Leimkuhler, *Relaxation techniques in multibody dynamics*, Trans. Canadian Soc. Mech. Engrg., to appear, 1993.

[6] B. Leimkuhler, U. Miekkala, and O. Nevanlinna, *Waveform relaxation for linear* RC-*Circuits*, IMPACT 3 (1991), pp. 123–145.

[7] B. Leimkuhler and A. Ruehli, *Rapid convergence of waveform relaxation*, J. Appl. Numer. Math., to appear, 1992.

[8] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, *The waveform relaxation method for time-domain analysis of large scale integrated circuits*, IEEE CAD for IC Systems, 3 (1982), pp. 131–145.

[9] U. Miekkala, *Dynamic iteration methods applied to linear* DAE *systems*, J. Comput. Appl. Math., 25 (1989), pp. 133–151.

[10] U. Miekkala and O. Nevanlinna, *Convergence of dynamic iteration methods for initial value problems*, SIAM J. Sci. Statist. Comput., 4 (1987), pp. 459–482.

[11] ———, *Sets of convergence and stability regions*, BIT, 27 (1987), pp. 554–584.

[12] U. Miekkala, O. Nevanlinna, and A. L. Ruehli, *Convergence and circuit partitioning aspects for waveform relaxation*, Research Rep., IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, 1990.

[13] O. Nevanlinna, *Remarks on Picard Lindelöf iteration*, BIT, 29 (1989), pp. 328–346.

[14] M. Reichelt, J. White, and J. Allen, *Waveform relaxation for transient simulation of two-dimensional* MOS *Devices*, Proc. IEEE Internat. Conf. Computer-Aided Design, 1989.

[15] A. Ruehli, private communication, 1992.

[16] A. L. Sangiovanni-Vincentelli and J. K. White, *Partitioning algorithms and parallel implementation of waveform relaxation algorithms for circuit simulation*, Proc. Internat. Symp. Circuits and Systems, Kyoto, Japan, 1985.

[17] S. M. Sharuz and F. Ma, *Approximate decoupling of the equations of motion of large flexible structures*, paper presented at American Control Conf., Pittsburgh, PA, 1989.

[18] K. Singal and J. Vlach, *Formulation of circuit equations*, in Circuit Analysis, Simulation and Design, Vol. 3, A. E. Ruehli, ed., Elsevier, North-Holland, Amsterdam, 1986.

[19] J. K. White and A. L. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of* VLSI *Circuits*, Kluwer Academic Publishers, Norwell, MA, 1986.

[20] D. M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 890–913, July 1993

PERFORMANCE ENHANCEMENTS AND PARALLEL ALGORITHMS FOR TWO MULTILEVEL PRECONDITIONERS*

H. C. ELMAN† AND XIAN-ZHONG GUO‡

Abstract. Several new variants of the hierarchical basis (HB) preconditioner and Bramble, Pasciak, and Xu's multilevel preconditioner (BPX) are presented and studied, and new parallel algorithms are introduced for both methodologies. It is shown that the performance of both preconditioners is improved by not solving the linear system associated with the initial (coarsest) grid, which need not be a trivial grid. For a class of problems consisting of anisotropic and "piecewise anisotropic" problems, a grid-generation strategy that relates the multilevel preconditioners to the underlying operator is developed, and it is demonstrated that this strategy is effective in terms of both iteration counts and elapsed time. In addition, a new parallel algorithm is presented for the BPX preconditioner that is as efficient as parallel algorithms for the HB preconditioner, requiring $O(j)$ parallel steps for problems with j levels; and a parallel algorithm is presented for the HB preconditioner that requires $O(\lceil \log_2 j \rceil)$ parallel steps. Numerical results on a Connection Machine are reported.

Key words. hierarchical basis, multilevel preconditioners, anisotropic problems, parallel algorithms, Connection Machine

AMS subject classifications. 65F10, 65F50, 65N20, 65W05

1. Introduction.

Consider the model elliptic partial differential equation

$$\text{(1)} \qquad \begin{cases} -\sum_{i=1}^{d} \frac{\partial}{\partial x_i}(a_i(x)\frac{\partial u}{\partial x_i}) = f \quad \text{on} \quad \Omega \subset R^d, \\ u\,|_{\partial\Omega} = 0, \end{cases}$$

where $\{a_i(x)\}$ satisfy $0 < m \leq a_i(x) \leq M$ for positive constants m and M, independent of $x \in \Omega$ and i. Discretization of (1) by finite difference methods or finite element methods leads to a linear system of equations

$$\text{(2)} \qquad A\underline{x} = \underline{b},$$

where A is the stiffness matrix, which is symmetric and positive definite. We are interested in solving the system (2) using the preconditioned conjugate gradient method (PCG), for which it is known that an upper bound on the number of iterations for convergence is proportional to the square root of the condition number of the preconditioned matrix; see, e.g., [6]. We study in this paper two multilevel preconditioners, the hierarchical basis (HB) preconditioner [10], [17] and the Bramble–Pasciak–Xu (BPX) multilevel preconditioner [3], [13]. HB is known to be nearly optimal for two-dimensional boundary value problems, producing condition numbers that grow like $O((\log h^{-1})^2)$ [17]. BPX is proved in [3] to be nearly optimal, with condition numbers growing like $O((\log h^{-1})^2)$ for both two-dimensional and higher problems. (Recently Oswald [11] proved that the condition number of the BPX-preconditioned system can actually be bounded independently of the level numbers.)

*Received by the editors October 28, 1991; accepted for publication (in revised form) June 12, 1992. This work 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 on a Connection Machine was provided by the Institute for Advanced Computer Studies, University of Maryland.

†Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742.

‡Department of Mathematics, University of Maryland, College Park, Maryland 20742. Present address, Engineering Software Research and Development, Inc., 7750 Clayton Rd., Suite 204, St. Louis, Missouri 63117.

Both of these preconditioners are defined in terms of an initial grid, and both require the solution (at each step of PCG) of a linear system with coefficient matrix A_0, where A_0 is the stiffness matrix associated with the initial grid. Unless the initial grid is coarse, the cost of the direct solution may be expensive, especially in three dimensions. To avoid the expense of the direct solution, one may replace A_0 by an appropriate preconditioner, as suggested in [3], [13], and [15]. In this paper, we consider variants of HB and BPX preconditioners in which A_0 is simply replaced by the identity matrix. We refer to these variants as incomplete multilevel preconditioners, and we refer to the multilevel preconditioners that apply A_0^{-1} as complete multilevel preconditioners. We demonstrate empirically that the incomplete versions actually converge at about the same rate as or faster than the complete versions, as long as the initial grid is not very fine. Moreover, the incomplete versions appear to be superior in terms of total cost for a fairly large range of nontrivial initial triangulations.

The HB method is defined only in terms of meshes, and not in terms of the differential operators. Because of this, the performance of the preconditioner may vary dramatically depending on the underlying operator. (This phenomenon can be observed from numerical experiments in this paper.) One method developed to make the HB preconditioner more closely related to the differential operator is to combine diagonal scaling with the hierarchical basis method [7]. Diagonal scaling is economical but it is ineffective for highly anisotropic problems. Another approach is the hierarchical basis multigrid method (HBMG), developed by Bank, Dupont, and Yserentant [2], which combines the multigrid idea with the hierarchical basis. HBMG is sophisticated but appears to be expensive. Aimed at a class of problems comprised by anisotropic and "piecewise anisotropic" problems, we develop another technique to relate the multilevel preconditioners to the underlying operator. Our idea is to choose the initial grid according to properties of the underlying operator coefficients, i.e., the initial grid is chosen in such way that data movement associated with the multilevel preconditioners mimics the physical process modeled by the differential equation. This technique does not need any extra work and it is very effective in terms of both iteration counts and elapsed time, and it works well for both the HB and BPX methods.

Implementations of the two multilevel preconditioners on parallel computers have been studied in various references. The hierarchical basis preconditioner has been considered on a shared memory Flexible-32 [1], [9], a Cray X-MP/24 [10], an NYU Ultracomputer Prototype [7], and a Connection Machine [4], [12]. An implementation of the BPX preconditioner has been considered on an iPSC/2 in [16]. The implementations of the HB preconditioner use a parallel version of the algorithm given in [17]. Essentially, this algorithm parallelizes all operations on each level, and different levels are treated serially. Thus, the cost of this parallel algorithm is $O(j)$, where j is the number of levels. Based on this parallel algorithm for the HB preconditioner, we will present a parallel algorithm for the BPX preconditioner that also costs $O(j)$. In two dimensions, this parallel algorithm makes the cost of BPX preconditioner exactly the same as that of the HB preconditioning equation. In addition, we will present in this paper a new parallel algorithm for the HB preconditioner that is more efficient than the standard version.

The paper is organized as follows. Section 2 introduces the HB and BPX preconditioners and briefly discusses their relationship from a matrix point of view. Section 3 shows the performance of the incomplete versions of the two multilevel preconditioners. Section 4 describes the effect of diagonal scaling and introduces a new efficient technique that uses the initial triangulation to relate the multilevel preconditioners more closely to the underlying partial differential operator. Section 5 examines parallel algorithms for the multilevel preconditioners.

All the numerical experiments described in this paper were done on a Connection Machine-2 (CM2). Computations were done in bit serial double precision with a zero initial guess, and the stopping criterion for convergence was

$$\| r^{(k)} \|_2 < 10^{-6} \times \| r^{(0)} \|_2,$$

where $r^{(k)}$ is the residual at the kth iteration and $\| \bullet \|_2$ is the vector Euclidean norm. Unless otherwise specified, the CM times reported in this paper came from a CM2 with 8,192 physical processors, and the HB preconditioner was implemented using Algorithm I of §5.

The following test problems on a unit square (for two-dimensional problems) or on a unit cube (for three-dimensional problems) will be used for our numerical experiments throughout the paper:

Problem 2-1. $d = 2$, $a_1 = a_2 = 1$, $u = x_1(x_1 - 1)x_2(x_2 - 1)$.

Problem 2-2. $d = 2$, $a_1 = 100, a_2 = 1$, $u = x_1(x_1 - 1)x_2(x_2 - 1)$.

Problem 2-3. $d = 2$, $f = 1$,

$$a_1 = \begin{cases} 100 & \text{if } 0 \le x_1 < 0.5, \\ 1 & \text{if } 0.5 \le x_1 \le 1; \end{cases} \qquad a_2 = \begin{cases} 1 & \text{if } 0 \le x_1 < 0.5, \\ 100 & \text{if } 0.5 \le x_1 \le 1. \end{cases}$$

Problem 2-4. $d = 2$, $f = 1$,

$$a_1 = \begin{cases} 1 & \text{if } 0.25 \le x_1, x_2 < 0.75, \\ 100 & \text{otherwise}; \end{cases} \qquad a_2 = \begin{cases} 100 & \text{if } 0.25 \le x_1, x_2 < 0.75, \\ 1 & \text{otherwise}. \end{cases}$$

Problem 3-1. $d = 3$, $a_1 = a_2 = a_3 = 1$, $u = x_1(x_1 - 1)x_2(x_2 - 1)x_3(x_3 - 1)$.

Problem 3-2. $d = 3$, $a_1 = a_2 = a_3 = 0.01 + x_1^2 + x_2^2 + x_3^2$, $u = x_1(x_1 - 1)x_2(x_2 - 1)x_3(x_3 - 1)$.

Problem 3-3. $d = 3$, $f = 1$,

$$a_1 = \begin{cases} 100 & \text{for } x_1 < 0.5, \\ 50.5 & \text{for } x_1 = 0.5, \\ 1 & \text{for } x_1 > 0.5; \end{cases} \qquad a_2 = a_3 = \begin{cases} 1 & \text{for } x_1 < 0.5, \\ 50.5 & \text{for } x_1 = 0.5, \\ 100 & \text{for } x_1 > 0.5. \end{cases}$$

Problems 2-2, 2-3, 2-4, and 3-3 are used to examine the effects of anisotropy and discontinuous coefficients. Problem 3-2 is used to examine the effects of variable coefficients.

2. Two multilevel preconditioners.

In this section, we introduce the HB and BPX preconditioners and briefly discuss their relationship from a matrix point of view. Let $\mathcal{T}_0, \ldots, \mathcal{T}_j$ denote a nested family of triangulations of Ω, as defined in [17]. For simplicity, assume that the finite element method with linear elements is used. Let M_k denote the finite dimensional space of all continuous piecewise linear functions on $\mathcal{T}_k$. Let $\mathcal{N}_k$ denote the set of unknown nodes (vertices) of $\mathcal{T}_k$ (e.g., the set of all interior nodes for Dirichlet boundary conditions), let $\mathcal{U}_k$ denote the set of new unknown nodes introduced at level k, and let n_k denote the number of nodes in $\mathcal{N}_k$. Clearly,

$$\mathcal{N}_k \subset \mathcal{N}_{k+1}, \qquad n_k \le n_{k+1},$$
$$\mathcal{U}_k = \mathcal{N}_k \setminus \mathcal{N}_{k-1}\ (k > 0), \qquad \mathcal{U}_0 = \mathcal{N}_0.$$

Let $H_0^1(\Omega)$ denote the usual Sobolev space; see, e.g., [5], and let $\mathcal{I}_k$: $H_0^1(\Omega) \to M_k$ denote the interpolation operator such that

$$\mathcal{I}_k u(x) = u(x) \quad \forall x \in \mathcal{N}_k.$$

Define the nodal basis of M_k as $\{\phi_l^{(k)}\}_{l \in \mathcal{N}_k}$, where $\phi_l^{(k)}$ is a piecewise linear polynomial on $\mathcal{T}_k$ and has value 1 at node l and value 0 at all other nodes of $\mathcal{T}_k$. The hierarchical basis of M_k is defined as $\bigcup_{s=0}^{k} \{\phi_l^{(s)}\}_{l \in \mathcal{U}_s}$.

Let A denote the global stiffness matrix derived from the nodal basis, let S denote the transformation matrix from the hierarchical basis to the nodal basis, and let A_0 denote the nodal basis discretization matrix with respect to $\mathcal{T}_0$. Assume that $A_0 = L_0 L_0^T$ is the Cholesky factorization. It is known [10], [17] that

$$\kappa(M^{-1}A) \le \begin{cases} C(j^2) & \text{in two dimensions,} \\ C(2^j) & \text{in three dimensions,} \end{cases} \tag{3}$$

with the preconditioner defined by

$$M^{-1} = S \tilde{L}_0^{-T} D_d^{-1} \tilde{L}_0^{-1} S^T, \tag{4}$$

where $\kappa(\cdot)$ stands for the condition number,

$$\tilde{L}_0 = \begin{pmatrix} L_0 & 0 \\ 0 & I \end{pmatrix},$$

and

$$D_d = \begin{pmatrix} h_0^{d-2} I_0 & & & & \\ & h_1^{d-2} I_1 & & & \\ & & h_2^{d-2} I_2 & & \\ & & & \ddots & \\ & & & & h_j^{d-2} I_j \end{pmatrix}.$$

Here, we order all nodes in a hierarchical ordering, i.e., we order the nodes in $\mathcal{U}_0$ first, followed by those in $\mathcal{U}_1$, etc. I_k is the identity matrix of order $n_k - n_{k-1}$, with $n_{-1} \equiv 0$.

Evaluating $L_0^{-T} v^{(0)}$ and $L_0^{-1} v^{(0)}$, i.e., solving two triangular systems $L_0^T x^{(0)} = v^{(0)}$ and $L_0 x^{(0)} = v^{(0)}$, could be expensive except for a very coarse initial triangulation $\mathcal{T}_0$. To avoid solving these systems, we define a new preconditioner by omitting L_0 and L_0^T, giving

$$M_I^{-1} = S D_d^{-1} S^T. \tag{5}$$

We refer to the preconditioner defined by (4) as the complete hierarchical basis (CHB) preconditioner and that defined by (5) as the incomplete hierarchical basis (IHB) preconditioner.

We can derive an upper bound on the condition number of the IHB-preconditioned system as follows. For any vector $x \neq 0$,

$$\frac{(Ax, x)}{(M_I x, x)} = \frac{(Ax, x)}{(Mx, x)} \frac{(Mx, x)}{(M_I x, x)},$$

where for vectors x and y, (x, y) is the usual Euclidean inner product. Thus

$$\begin{aligned}\lambda_{\max}(M_I^{-1}A) &= \max_{x\neq 0}\frac{(Ax,x)}{(M_Ix,x)}\\ &\leq \max_{x\neq 0}\frac{(Ax,x)}{(Mx,x)}\times\max_{x\neq 0}\frac{(Mx,x)}{(M_Ix,x)},\\ \lambda_{\min}(M_I^{-1}A) &= \min_{x\neq 0}\frac{(Ax,x)}{(M_Ix,x)}\\ &\geq \min_{x\neq 0}\frac{(Ax,x)}{(Mx,x)}\times\min_{x\neq 0}\frac{(Mx,x)}{(M_Ix,x)},\end{aligned}$$

and therefore

$$\kappa(M_I^{-1}A) = \frac{\lambda_{\max}(M_I^{-1}A)}{\lambda_{\min}(M_I^{-1}A)} \leq \kappa(M^{-1}A)\times\kappa(M_I^{-1}M).$$

By (3) and the fact that

$$\kappa(M_I^{-1}M) = \kappa(A_0) \leq Ch_0^{-2},$$

where h_0 is the minimum mesh size of $\mathcal{T}_0$, we get the following theorem.

THEOREM 2.1. *If M_I is defined by* (5), *then*

$$\kappa(M_I^{-1}A) \leq \begin{cases} C(h_0^{-2}j^2) & \textit{in two dimensions},\\ C(h_0^{-2}2^j) & \textit{in three dimensions},\end{cases} \tag{6}$$

where h_0 is the minimum mesh size of $\mathcal{T}_0$.

We next consider the BPX preconditioner, using the algebraic framework developed in [13] and [15]. Let T_k denote the representation matrix of the nodal basis $\{\phi_l^{(k)}\}_{l\in\mathcal{N}_k}$ in terms of the nodal basis $\{\phi_l^{(j)}\}_{l\in\mathcal{N}_j}$:

$$\begin{bmatrix}\phi_1^{(k)}\\ \vdots\\ \phi_{n_k}^{(k)}\end{bmatrix} = T_k\begin{bmatrix}\phi_1^{(j)}\\ \vdots\\ \phi_{n_j}^{(j)}\end{bmatrix}.$$

In matrix form, the BPX preconditioner is defined by [13]

$$M^{-1} = h_j^{2-d}I + \sum_{k=1}^{j-1} h_k^{2-d}T_k^TT_k + h_0^{2-d}T_0^TA_0^{-1}T_0, \tag{7}$$

where h_k is the mesh size of $\mathcal{T}_k$ and d is the number of dimensions. (We assume here that a uniform mesh is used; for locally refined meshes, see [3] and [13].) It is shown in [3] and [13] that

$$\kappa(M^{-1}A) \leq \begin{cases} C(j^2) & \text{in general},\\ Cj & \text{for regular problems}.\end{cases} \tag{8}$$

More recently, it has been shown [11] that the condition number of the BPX-preconditioned matrix can actually be bounded independently of mesh size.

As above, one may avoid applying A_0^{-1} by using a modified preconditioner defined by

$$M_I^{-1} = h_j^{2-d} I + \sum_{k=0}^{j-1} h_k^{2-d} T_k^T T_k. \tag{9}$$

We refer to the preconditioner defined by (7) as the complete BPX (CBPX) preconditioner, and that defined by (9) as the incomplete BPX (IBPX) preconditioner. It is shown in [13] that

$$\kappa(M_I^{-1}A) \le \begin{cases} C(h_0^{-2} + j)j & \text{in general,} \\ C(h_0^{-2} j) & \text{for regular problems.} \end{cases} \tag{10}$$

As in the proof of Theorem 2.1, we can also show that the IBPX-preconditioned matrix has condition number bounded by Ch_0^{-2}, based on the results in [11].

The HB and BPX preconditioners are closely related to each other [13], [14], [15], [18]. Here, we briefly discuss the relation between them from a matrix point of view; see [15] for some related analysis. Let T_k^{k+1} be the matrix that represents the nodal basis $\{\phi_l^{(k)}\}_{l \in \mathcal{N}_k}$ in terms of the nodal basis $\{\phi_l^{(k+1)}\}_{l \in \mathcal{N}_{k+1}}$:

$$\begin{bmatrix} \phi_1^{(k)} \\ \vdots \\ \phi_{n_k}^{(k)} \end{bmatrix} = T_k^{k+1} \begin{bmatrix} \phi_1^{(k+1)} \\ \vdots \\ \phi_{n_{k+1}}^{(k+1)} \end{bmatrix}. \tag{11}$$

Clearly,

$$T_k = T_k^{k+1} \cdots T_{j-1}^{j}.$$

Define $\hat{S}_k = (T_{k-1}^k)^T$. Then (7) can be rewritten as [13]

$$\begin{aligned} M^{-1} &= h_j^{2-d} I + \sum_{k=2}^{j} h_{k-1}^{2-d} (\hat{S}_j \cdots \hat{S}_k)(\hat{S}_k^T \cdots \hat{S}_j^T) \\ &\quad + h_0^{2-d} \hat{S}_j \cdots \hat{S}_1 A_0^{-1} \hat{S}_1^T \cdots \hat{S}_j^T \end{aligned} \tag{12}$$

and (9) can be rewritten as

$$M_I^{-1} = h_j^{2-d} I + \sum_{k=1}^{j} h_{k-1}^{2-d} (\hat{S}_j \cdots \hat{S}_k)(\hat{S}_k^T \cdots \hat{S}_j^T). \tag{13}$$

Now consider the matrix S which transforms the hierarchical basis to the nodal basis. S can be factored as [17]

$$S = S_j \cdots S_1.$$

Given $u \in \mathcal{M}_j$, its nodal basis coefficients are $\{u_i = u(x_i) \mid x_i \in \mathcal{N}_j\}$ and its hierarchical basis coefficients are $\{(\mathcal{I}_k - \mathcal{I}_{k-1})u(x_i) \mid x_i \in \mathcal{U}_k, 0 \le k \le j\}$, where $\mathcal{I}_{-1} \equiv 0$. Let v

denote the vector of the hierarchical basis coefficients of u. Then Sv is the vector of the nodal basis coefficients of u. S_k describes the computation of $\{u_i \mid i \in \mathcal{U}_k\}$ from $\{(\mathcal{I}_k - \mathcal{I}_{k-1})u(x_i) \mid x_i \in \mathcal{U}_k\}$ and $\{u_i \mid i \in \mathcal{N}_{k-1}\}$. Thus, in order to evaluate the action of S_k, one first needs to evaluate the function $\mathcal{I}_{k-1}u$ at nodes from $\mathcal{U}_k$. This turns out to be a simple interpolation process [17]. Then one adds the values of $\mathcal{I}_k u - \mathcal{I}_{k-1}u$ at these nodes to the interpolated values, giving the values of u at nodes from $\mathcal{U}_k$. If we order the nodes in a hierarchical ordering, then for $1 \le k \le j$, S_k has the structure

$$S_k = \begin{pmatrix} I_0 & & & & & & \\ & \ddots & & & & & \\ & & I_{k-1} & & & & \\ R_0 & \cdots & R_{k-1} & I_k & & & \\ & & & & I_{k+1} & & \\ & & & & & \ddots & \\ & & & & & & I_j \end{pmatrix}. \tag{14}$$

In terms of entries of S_k, the interpolated values of $\mathcal{I}_{k-1}u$ at nodes from $\mathcal{U}_k$ are given by

$$\sum_{s=0}^{k-1} R_s \underline{u}_s, \tag{15}$$

where $\underline{u}_s$ is the vector of values of u at nodes from $\mathcal{U}_s$.

On the other hand, since $\mathcal{I}_{k-1}u \in \mathcal{M}_{k-1}$ and $\mathcal{I}_{k-1}u(x_i) = u(x_i) = u_i$ for $i \in \mathcal{N}_{k-1}$, we have

$$\mathcal{I}_{k-1}u = \sum_{i=1}^{n_{k-1}} u_i \phi_i^{(k-1)} = (\phi_1^{(k-1)}, \dots, \phi_{n_{k-1}}^{(k-1)}) \begin{bmatrix} u_1 \\ \vdots \\ u_{n_{k-1}} \end{bmatrix}.$$

From (11) and the equality $\hat{S}_k = (T_{k-1}^k)^T$, one obtains

$$(\phi_1^{(k-1)}, \dots, \phi_{n_{k-1}}^{(k-1)}) = (\phi_1^{(k)}, \dots, \phi_{n_k}^{(k)})\hat{S}_k.$$

Therefore

$$\mathcal{I}_{k-1}u = (\phi_1^{(k)}, \dots, \phi_{n_k}^{(k)})\hat{S}_k \begin{bmatrix} u_1 \\ \vdots \\ u_{n_{k-1}} \end{bmatrix},$$

and the values of $\mathcal{I}_{k-1}u$ at nodes from $\mathcal{N}_k$ are then given by

$$\hat{S}_k \begin{bmatrix} u_1 \\ \vdots \\ u_{n_{k-1}} \end{bmatrix}.$$

But $\mathcal{I}_{k-1}u(x_i) = u_i$ for $i \in \mathcal{N}_{k-1}$, so that $\hat{S}_k$ has the structure

$$\hat{S}_k = \begin{bmatrix} I_0 & & \\ & \ddots & \\ & & I_{k-1} \\ \hat{R}_0 & \cdots & \hat{R}_{k-1} \end{bmatrix}.$$

It is now easy to see that the interpolated values of $\mathcal{I}_{k-1}u$ at nodes from $\mathcal{U}_k$ are represented in terms of entries of $\hat{S}_k$ as

$$\sum_{s=0}^{k-1} \hat{R}_s \underline{u}_s. \tag{16}$$

Since (15) and (16) both represent the same interpolated values of $\mathcal{I}_{k-1}u$ at nodes from $\mathcal{U}_k$, it follows that

$$R_s = \hat{R}_s, \qquad s = 0, \ldots, k-1.$$

In other words, $\hat{S}_k$ is composed of the first k block columns and the first $k+1$ block rows of S_k:

$$\hat{S}_k = \begin{pmatrix} I_0 & & \\ & \ddots & \\ & & I_{k-1} \\ R_0 & \cdots & R_{k-1} \end{pmatrix}. \tag{17}$$

S_k is square and of size $n_j \times n_j$, whereas $\hat{S}_k$ is rectangular and of size $n_k \times n_{k-1}$.

3. Performance of the IHB and IBPX preconditioners. We now present the results of some numerical experiments that give us insight into the computational properties of the incomplete multilevel preconditioners. In our numerical experiments, the finite element method with linear elements is used for two-dimensional problems. The triangulation has the uniform mesh size $h = 1/n$, and the initial triangulation corresponds to a uniform grid with $h_0 = 1/n_0$, where $n_0 = n/2^j$ and j is the number of levels excluding the initial level. For simplicity of implementation, the standard seven-point centered difference scheme on a uniform mesh is used to discretize the three-dimensional problems, and our multilevel preconditioning matrices correspond to those derived from the finite element method with trilinear elements, except that A_0 is the matrix derived from the difference scheme. From the spectral equivalence of the standard seven-point centered difference operator and that derived from the finite element method with trilinear elements, we know that the HB and BPX preconditioners have the same convergence properties as those given in §2.

Table 1 shows the number of PCG iterations using the complete and incomplete versions of the HB and BPX preconditioners for Problem 2-1, where "-" stands for "data not available." Table 2 shows analogous results for Problem 3-1. We see that when the initial grid is not very fine, the incomplete versions of the preconditioners actually require about the same number of iterations (and fewer, in many cases) as the complete versions.

TABLE 1
Iteration counts for Problem 2-1 (*constant coefficients*).

j	n=32	n=64	n=128	n=256	n=32	n=64	n=128	n=256
	IHB				CHB			
7				74				78
6			61	70			64	85
5		49	57	67		51	70	75
4	38	45	55	70	40	57	62	59
3	35	43	58	92	44	48	47	—
2	33	47	81	138	36	36	34	—
1	39	67	122	214	23	22	21	—
0	43	88	178	358	1	1	1	1
j	n=32	n=64	n=128	n=256	n=32	n=64	n=128	n=256
	IBPX				CBPX			
7				28				28
6			27	28			26	29
5		24	26	31		24	26	31
4	22	24	29	46	21	24	29	31
3	21	26	42	80	22	26	28	—
2	23	39	74	144	23	24	24	—
1	34	66	128	243	19	19	18	—
0	43	88	178	358	1	1	1	1

TABLE 2
Iteration counts for Problem 3-1 (*constant coefficients*).

j	n=8	n=16	n=32	n=64	n=8	n=16	n=32	n=64
	IHB				CHB			
5				161				169
4			102	140			106	140
3		62	87	106		64	85	99
2	31	51	65	82	31	50	59	76
1	27	38	54	91	31	34	43	—
0	12	24	49	103	1	1	1	1
j	n=8	n=16	n=32	n=64	n=8	n=16	n=32	n=64
	IBPX				CBPX			
5				44				43
4			35	38			35	37
3		27	31	36		27	30	32
2	19	25	31	53	19	24	26	31
1	18	26	43	82	17	20	26	—
0	12	24	49	103	1	1	1	1

Taking into account the extra work for the complete versions to apply A_0^{-1}, particularly in three dimensions, we see that PCG with the IHB or IBPX preconditioner costs less than with its complete counterpart, at least when the initial grid is not very fine.

Note that the upper bounds of the IHB- or IBPX-preconditioned matrices in §2 increase as j decreases. However, from Tables 1 and 2, we see that as j decreases from its largest value (i.e., as we vary the initial grid from very coarse to slightly less coarse), the iteration counts for both the IHB and IBPX preconditioners essentially do not increase. Since the cost per PCG step decreases as j decreases, we conclude that better performance will be achieved when the initial grid is not as coarse as possible. This conclusion is further substantiated by Tables 3 and 4, which show the CPU times of PCG with the IHB and IBPX preconditioners for Problem 2-1 and Problem 3-1, respectively, with different numbers of levels. Here the CM times for the three-dimensional $64 \times 64 \times 64$ grid problem come from a CM2 with 16,384 (instead of 8,192) physical processors.

TABLE 3
CM time (seconds) for Problem 2-1.

j	n=32	n=64	n=128	n=256	n=32	n=64	n=128	n=256
	IHB				IBPX			
7				83.24				34.55
6			17.36	71.18			8.80	31.08
5		7.13	14.41	59.90		3.77	7.23	30.36
4	4.84	5.69	12.11	53.82	3.03	3.29	7.17	38.53
3	3.80	4.63	10.84	58.88	2.48	3.01	8.55	56.25
2	2.94	4.59	12.26	72.73	2.19	3.65	12.23	84.35
1	2.73	4.59	14.61	90.54	2.56	5.03	17.11	116.08

TABLE 4
CM time (seconds) for Problem 3-1.

j	n=8	n=16	n=32	n=64	n=8	n=16	n=32	n=64
	IHB				IBPX			
5				437.79				137.76
4			68.36	338.40			25.64	106.29
3		10.08	49.10	224.88		4.83	19.73	87.29
2	4.30	6.91	31.31	149.70	2.93	3.79	16.82	105.76
1	3.07	4.66	21.61	134.37	2.26	3.16	18.24	126.88

Remark. We saw the same general behavior for problems with variable and discontinuous coefficients, and also with random initial guesses, although only the results of the Laplacian with a zero initial guess are reported here.

4. Relating multilevel preconditioners to the underlying operator.

The HB and BPX preconditioners under consideration are defined independently of the underlying differential operator, and their effectiveness may vary dramatically depending on this operator. In this section, we examine two techniques for relating multilevel preconditioners to the differential operator.

One strategy is to use a diagonal scaling, as suggested for the HB preconditioner by Greenbaum, Li, and Chao [7]. Combining a diagonal scaling with IHB leads to the preconditioner

$$M^{-1} = SD^{-1}S^T, \tag{18}$$

where D is a diagonal mattrix. One choice for D is $D = \operatorname{diag}(\hat{A})$, where $\hat{A}$ is the global discretization matrix derived from the hierarchical basis. A drawback of this choice is that the matrix $\hat{A}$ is a preconditioned matrix that is typically not explicitly formed. An alternative that avoids the cost of computing $\operatorname{diag}(\hat{A})$ is the choice $D = \operatorname{diag}(A)$, as suggested in [7]. Numerical experiments with these two choices for D produce almost identical iteration counts for discrete partial differential operators with continuous coefficients of the partial differential equations, as reported in [7] for two-dimensional problems with the complete hierarchical basis diagonal scaled preconditioner. Therefore, we will only use $D = \operatorname{diag}(A)$ for diagonal scaling. We refer to (18) with $D = \operatorname{diag}(A)$ as the incomplete hierarchical basis diagonal (IHBD) scaled preconditioner. One can also combine diagonal scaling with BPX [3], [13], [15], [19], and define the incomplete BPX diagonal (IBPXD) scaled preconditioner by

$$M^{-1} = h_j^{2-d} D_j^{-1} + \sum_{k=1}^{j} h_k^{2-d} (\hat{S}_k \cdots \hat{S}_1) D_k^{-1} (\hat{S}_1^T \cdots \hat{S}_k^T),$$

where D_k is the submatrix of diag(A) corresponding to the nodes from $\mathcal{N}_k$. This can be thought of as a Jacobi preconditioner at level k; cf. [15] for generalizations of this idea that more closely relate the preconditioner to the underlying operator.

Table 5 reports the iteration counts of PCG with IBPX and IBPXD preconditioners of all possible levels for Problem 3-2, and Table 6 reports some of the corresponding CPU times, together with the CPU time of red-black ICCG(0) [8] for comparison. In Table 6, CGD stands for PCG with the preconditioner $D =$ diag(A), and the parallel algorithm developed later in §5 is used for both the IBPX and IBPXD preconditioners. Tables 5 and 6 suggest that the diagonal scaled multilevel preconditioners work well for the problems with variable coefficients. We also see from Table 6 that with the diagonal scaled multilevel preconditioners, better performance is achieved when the initial grid is not as coarse as possible, as we observed with the IHB and IBPX preconditioners in §3.

TABLE 5
Iteration counts for Problem 3-2 (*variable coefficients*).

j	n=8	n=16	n=32	n=64	n=8	n=16	n=32	n=64
	IBPX				IBPXD			
5				204				44
4			151	204			35	39
3		91	152	213		28	31	41
2	45	91	160	247	20	25	36	65
1	45	99	187	333	19	31	57	112
0	36	91	204	429	20	40	83	172

TABLE 6
Problem 3-2. *IBPXD*(k) *means that initial grid is* $2^k \times 2^k \times 2^k$. $k = 1$ *corresponds the coarsest initial grid.* A 16*K CM*2 *is used for case* $n = 64$.

	Iteration counts				CM time (seconds)			
method	n=8	n=16	n=32	n=64	n=8	n=16	n=32	n=64
CGD	20	40	83	172	1.60	2.94	20.77	160.11
ICCG	14	25	48	96	2.01	3.33	22.33	164.36
IBPXD(1)	20	28	35	44	3.16	5.08	25.89	139.33
IBPXD(3)		31	36	41		3.78	19.72	99.92

Diagonal scaling has little or no effect on another class of problems, comprised by anisotropic problems, such as Problem 2-2, and "piecewise anisotropic" problems, such as Problems 2-3, 2-4, and 3-3. For instance, for any of these problems (with a uniform mesh), diag$(A) = cI$ for some scalar c and IHBD (or IBPXD) reduces to IHB (or IBPX). We now describe a technique, aimed at this class of problems, to relate the multilevel preconditioners more closely to the underlying operator. Note that the solution of the elliptic partial differential equation (1) can be interpreted as a distribution of heat, where the coefficients $\{a_i\}$ are the conducting coefficients. Thus, for example in two dimensions, if $a_1 > a_2$, then heat conducts more rapidly in the horizontal direction than in the vertical direction. We would like our computational algorithm to mimic the physical process, by patterning data movement after the physics. The initial triangulation used for the multilevel preconditioners provides a natural vehicle for achieving this.

For an example, assume that the discretization is made on a 4×4 grid, as in the top picture of Fig. 1. Three possible choices for the initial grid are shown in the second row of Fig. 1, and they give rise to the three partitionings of the nodes shown in the third row of Fig. 1. Assume that $x_{I1(i)}, x_{I2(i)} \in N_0$ are the two nodes that are directly connected

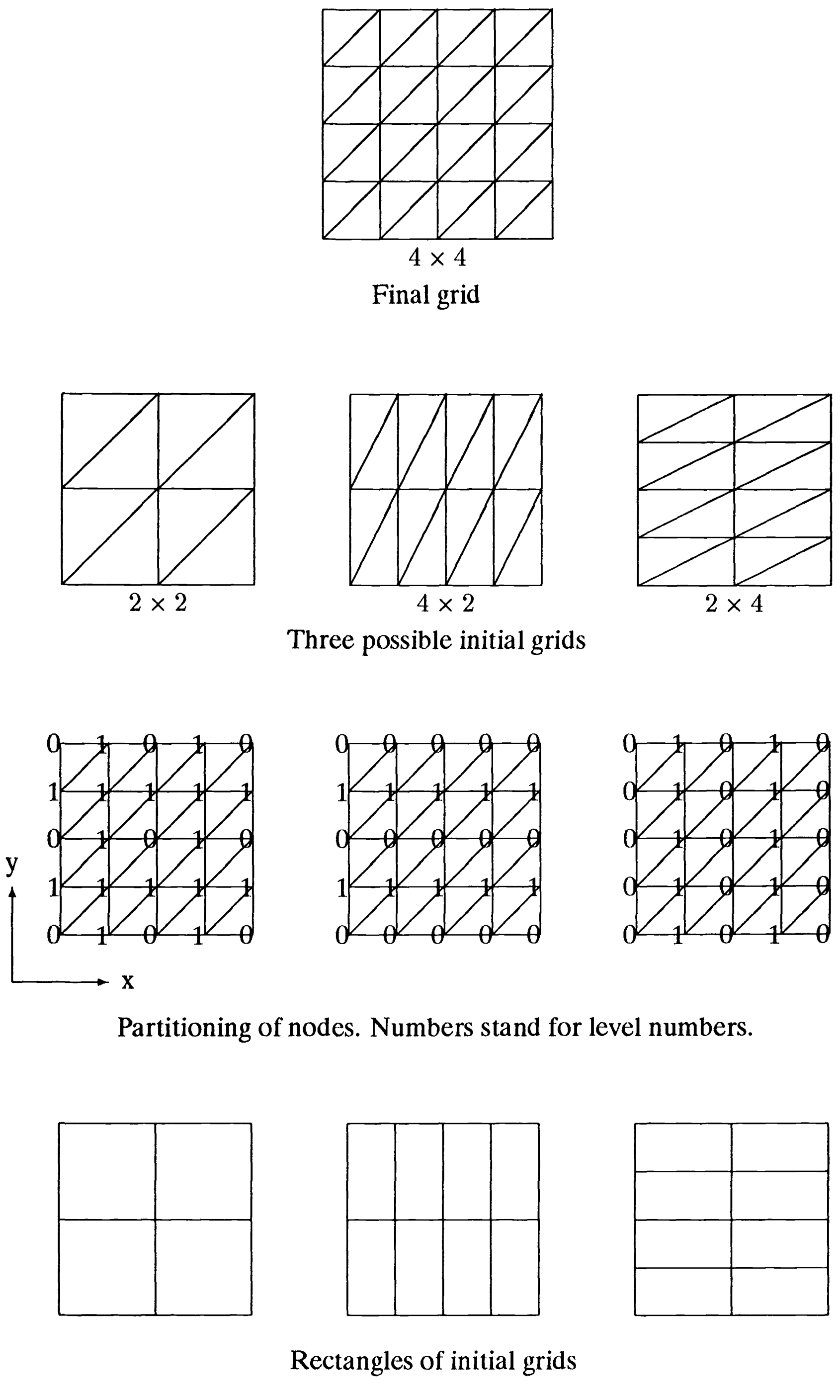

4 × 4
Final grid
2 × 2
4 × 2
2 × 4
Three possible initial grids
y
x
Partitioning of nodes. Numbers stand for level numbers.
Rectangles of initial grids

FIG. 1. *Three refinement strategies leading to a* 4×4 *final grid.*

to the node $x_i \in \mathcal{U}_1$, and that the data at node x_i is stored in $X(i)$. When we apply the HB preconditioner with a uniform mesh, the following algorithm may be used [17]:

```
for $i \in \mathcal{U}_1$
    $X(I1(i) = X(I1(i)) + X(i)/2$
    $X(I2(i) = X(I2(i)) + X(i)/2$
next $i$
for $i \in \mathcal{U}_1$
    $X(i) = X(i) + (X(I1(i)) + X(I2(i)))/2$
next $i$
```

It is useful to take the point of view that each node x_i corresponds to a processor, processor i, on a parallel computer. Thus, to apply the HB preconditioner, processor i needs to communicate with processors $I1(i)$ and $I2(i)$, to send some data to processors $I1(i)$ and $I2(i)$ and then to get some data from them. For the partitioning on the left in the third row of Fig. 1 (derived from an initial 2×2 grid), node (1,1) will communicate with its NE and SW neighbors, node (0,1) with its N and S neighbors, and node (1,0) with its E and W neighbors; for the partitioning in the center (derived from an initial 4×2 grid), every node from level 1 will communicate only with its N and S neighbors; and finally for the partitioning on the right (derived from an initial 2×4 grid), every node from level 1 will communicate only with its E and W neighbors. Therefore, the initial 2×2 grid results in data movement at the same rate in the horizontal and vertical directions, the initial 4×2 grid results in more rapid data movement in the vertical direction, and the initial 2×4 grid results in more rapid data movement in the horizontal direction.

Note that the initial 4×2 and 2×4 grids result in nonnested sequences of finite element subspaces $\{\mathcal{M}_k\}_{k=0}^{1}$, i.e., $\mathcal{M}_0 \not\subset \mathcal{M}_1$, because some triangles at level 0 cannot be expressed as unions of triangles at level 1. However, each rectangle of the initial triangulation can be expressed as a union of some rectangles of the next level triangulation, see the last row of Fig. 1. The analysis of the HB and BPX preconditioners applies only for the nested case [3], [10], [13], [17]. Nevertheless, all analysis is based on elementwise arguments, i.e., all inequalities and equalities are proven on each element and the sum over all elements is taken to get the results on the domain. Replacing triangle by rectangle, one can generalize the analysis of HB and BPX [3], [10], [13], [17] in a straightforward manner to cover the nonnested case described above.

As an example of the use of such initial triangulations, consider Problem 2-2. Table 7 shows the iteration counts for various initial grids. The results indicate that the initial 2×2 grid does not give the best performance among all possible initial triangulations. In this case, an initial 2×16 grid gives the best or almost best performance for both IBPX and IHB. The data of Table 7 typifies our experiments. In general, assume both a_1 and a_2 are constant and without loss of generality assume $a_2 > a_1$. Numerical experiments indicate that an initial $2 \times m_0$ grid will be close to optimal when $(m_0/2)^2$ is as close to a_2/a_1 as possible.

TABLE 7
Iteration counts on grid 256×256 *for Problem* 2-2.

Initial grid	2×2	2×4	2×8	2×16	2×32	2×64	2×128	2×256
IHB	159	98	64	55	51	55	76	121
IBPX	144	75	47	33	39	51	66	95

This idea can also be generalized to piecewise anisotropic problems. Consider Problem 2-3 first; see Fig. 2. For this problem, the discussion above suggests that the initial grid on the left half subregion should be finer in the y-direction than in the x-direction, and the opposite should hold on the right half subregion. Suppose again that the discretization is on a 4×4 grid. One way to choose such an initial triangulation is so that $(h_x^{(0)}, h_y^{(0)}) = (1/2, 1/4)$ on the left half region and $(h_x^{(0)}, h_y^{(0)}) = (1/4, 1/2)$ on the right half region, where $h_x^{(0)}$ and $h_y^{(0)}$ are the step sizes of the corresponding initial grid in the x- and y-directions, respectively; see Fig. 2. (The initial triangulation is designed in such a way that conforming finite element methods can be applied; see, e.g., [5].)

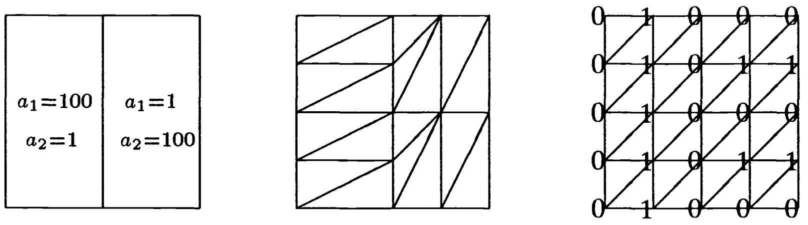


FIG. 2. *Left: Problem* 2-3. *Center: initial grid. Right: partitioning of the nodes.*

Table 8 reports the iteration counts of PCG with the IHB and IBPX preconditioners for Problem 2-3, where initial grid 2×2 means that $(h_x^{(0)}, h_y^{(0)}) = (1/2, 1/2)$ on both the left and right half regions, and initial grid $2 \times 16, 16 \times 2$ means that $(h_x^{(0)}, h_y^{(0)}) = (1/2, 1/16)$ on the left half subregion and $(h_x^{(0)}, h_y^{(0)}) = (1/16, 1/2)$ on the right half subregion.

TABLE 8
Iteration counts for Problem 2-3.

Initial grid	n=32	n=64	n=128	n=256	n=32	n=64	n=128	n=256
	IHB				IBPX			
2×2	112	136	152	169	114	143	160	175
$2 \times 16, 16 \times 2$	30	39	50	61	41	50	56	63

We can apply our technique to Problem 2-4 (Fig. 3) in a similar way. We will choose the initial triangulation such that the corresponding initial grid is finer in the x-direction on the inner subregion and it is finer in y-direction on the outer subregion. Figure 4 shows one such initial triangulation, together with the refined ones, for an 8×8 grid. Numerical results for various initial grids are reported in Table 9, where initial grid 2×2 means that $(h_x^{(0)}, h_y^{(0)}) = (1/2, 1/2)$ on both inner and outer subregions, and initial grid $2 \times 16, 16 \times 2$ means that $(h_x^{(0)}, h_y^{(0)}) = (1/2, 1/16)$ on the outer subregion and $(h_x^{(0)}, h_y^{(0)}) = (1/16, 1/2)$ on the inner subregion.

TABLE 9
Iteration counts for Problem 2-4.

Initial grid	n=32	n=64	n=128	n=256	n=32	n=64	n=128	n=256
	IHB				IBPX			
2×2	119	148	168	191	117	151	172	187
$2 \times 16, 16 \times 2$	45	51	61	75	44	59	70	76

The generalization of this technique to higher dimensional problems is straightforward. Table 10 shows the iteration counts for a three-dimensional problem, Problem 3-3, where initial grid $2 \times 2 \times 2$ means that $(h_x^{(0)}, h_y^{(0)}, h_z^{(0)}) = (1/2, 1/2, 1/2)$ on the whole region, and initial grid $2 \times 8 \times 8, 8 \times 8 \times 2$ means that $(h_x^{(0)}, h_y^{(0)}, h_z^{(0)}) = (1/2, 1/8, 1/8)$

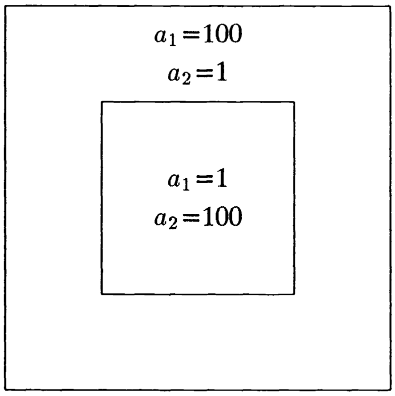


FIG. 3. *Problem* 2-4.

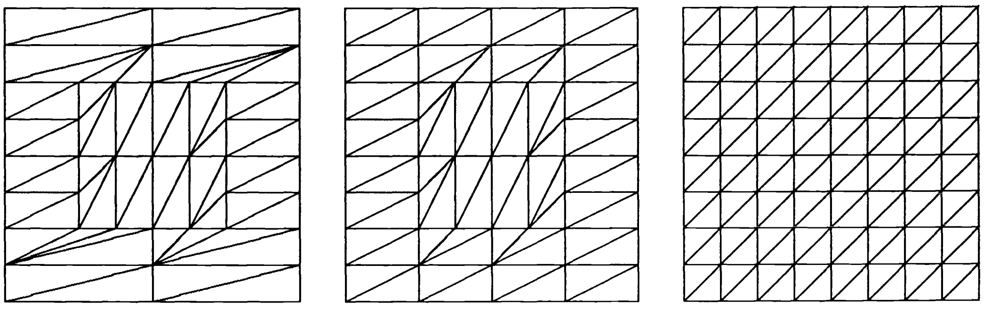


FIG. 4. *Sequence of triangulations for Problem* 2-4.

on the subregion $x_1 < 0.5$, and $(h_x^{(0)}, h_y^{(0)}, h_z^{(0)}) = (1/8, 1/8, 1/2)$ on the subregion $0.5 < x_1$.

TABLE 10
Iteration counts of IBPX-preconditioned CG for Problem 3-3.

Initial grid	n=8	n=16	n=32	n=64
$2\times2\times2$	39	82	146	201
$2\times8\times8, 8\times8\times2$	24	47	67	90

The new technique has the feature that it does not need any extra work and the implementation of the technique does not create any extra difficulty. For example, Table 11 reports iteration counts and CPU times for PCG with various preconditioners for Problem 2-4. We see from this table that our technique does work well for piecewise anisotropic problems. The technique can be generalized to problems with variable coefficients by first freezing the coefficients on each element of the initial triangulation, i.e., defining some constant value of the coefficients on each element, and then applying the technique to the corresponding stiffness matrix.

5. Parallel algorithms for multilevel preconditioners. In this section, we discuss parallel algorithms for the IHB and IBPX preconditioners. For simplicity, all algorithms

TABLE 11

Results for Problem 2-4. *IHB*-1 (*IBPX*-1) *is IHB* (*IBPX*) *with initial grid* 2×2 *and IHB-2* (*IBPX-2*) *is IHB* (*IBPX*) *with such initial grid that* $(h_x^{(0)}, h_y^{(0)}) = (1/16, 1/2)$ *on the inner subregion and* $(h_x^{(0)}, h_y^{(0)}) = (1/2, 1/16)$ *on the outer subregion.*

	Iteration counts				CM time (seconds)			
Method	n=32	n=64	n=128	n=256	n=32	n=64	n=128	n=256
CG	134	287	607	1248	6.28	13.29	52.21	397.09
ICCG	69	144	305	625	6.41	13.21	53.40	397.59
IHB-1	119	148	168	191	18.64	25.95	58.43	267.73
IHB-2	45	51	61	75	7.74	9.97	21.14	110.14
IBPX-1	117	151	172	187	17.33	26.41	61.26	260.85
IBPX-2	44	59	70	76	6.78	10.11	24.24	111.97

are described in detail only for one-dimensional problems; some issues for higher dimensional problems are outlined. Since each node corresponds to a unique index, we do not distinguish a node x_i from its index i.

With the IHB preconditioner, the core of solving the preconditioning equation is to compute $S^T v$ and Sv for an arbitrary vector v. Suppose the hierarchical basis coefficients are stored in a vector X. Given a node $x_i \in \mathcal{U}_k$, let $x_{I1(i)} \equiv \max\{x \mid x \in \mathcal{N}_{k-1},\ x < x_i\}$ and $x_{I2(i)} \equiv \min\{x \mid x \in \mathcal{N}_{k-1},\ x_i < x\}$. We refer to $x_{I1(i)}$ and $x_{I2(i)}$ as the parents of x_i and to x_i as a child of its parents. Assume that a uniform mesh is used. The standard parallel algorithms for computing Sv and $S^T v$ are the following straightforward variants of the serial algorithms of [17]. Here and in the sequel, we are assuming that there are as many processors as grid points.

Parallel Algorithm I Sv:

```
1  for $k = 1$ to $j$, do
2      for all $i \in \mathcal{U}_k$
3          $X(i) = X(i) + (X(I1(i)) + X(I2(i)))/2$
4      all $i$ in parallel
5  next $k$
```

Parallel Algorithm I $S^T v$:

```
1  for $k = j$ to 1, do
2      for all $i \in \mathcal{U}_k$, do
3          $X(I1(i)) = X(I1(i)) + X(i)/2$
4          $X(I2(i)) = X(I2(i)) + X(i)/2$
5      all $i$ in parallel
6  next $k$
```

Lines 2–4 of the algorithm for Sv compute the action of S_k, and lines 2–5 of the algorithm for $S^T v$ compute the action of S_k^T, where S_k $(k = 1, \ldots, j)$ are the factors of S defined in §2. Figures 5 and 6 show two symbolic representations of these algorithms. Figure 5 shows the flow of data between processors during the course of the computation for $j = 3$; arrows indicate the required interprocessor communication. In Fig. 6, the column associated with $\mathcal{U}_k$ contains a compact representation of the contents of all processors handling nodes in $\mathcal{U}_k$, where n/m means that all processors handling $x \in \mathcal{U}_k$ now contain $(\mathcal{I}_n u - \mathcal{I}_m u)(x)$. For example, after step 2, all processors handling $x \in \mathcal{U}_2$ contain $\mathcal{I}_2 u(x)$, whereas for $k \geq 3$, all processors handling $x \in \mathcal{U}_k$ contain $(\mathcal{I}_k u - \mathcal{I}_{k-1} u)(x)$. It is obvious that both Sv and $S^T v$ require j parallel steps and each step costs $O(1)$. Thus, this algorithm costs $O(j)$.

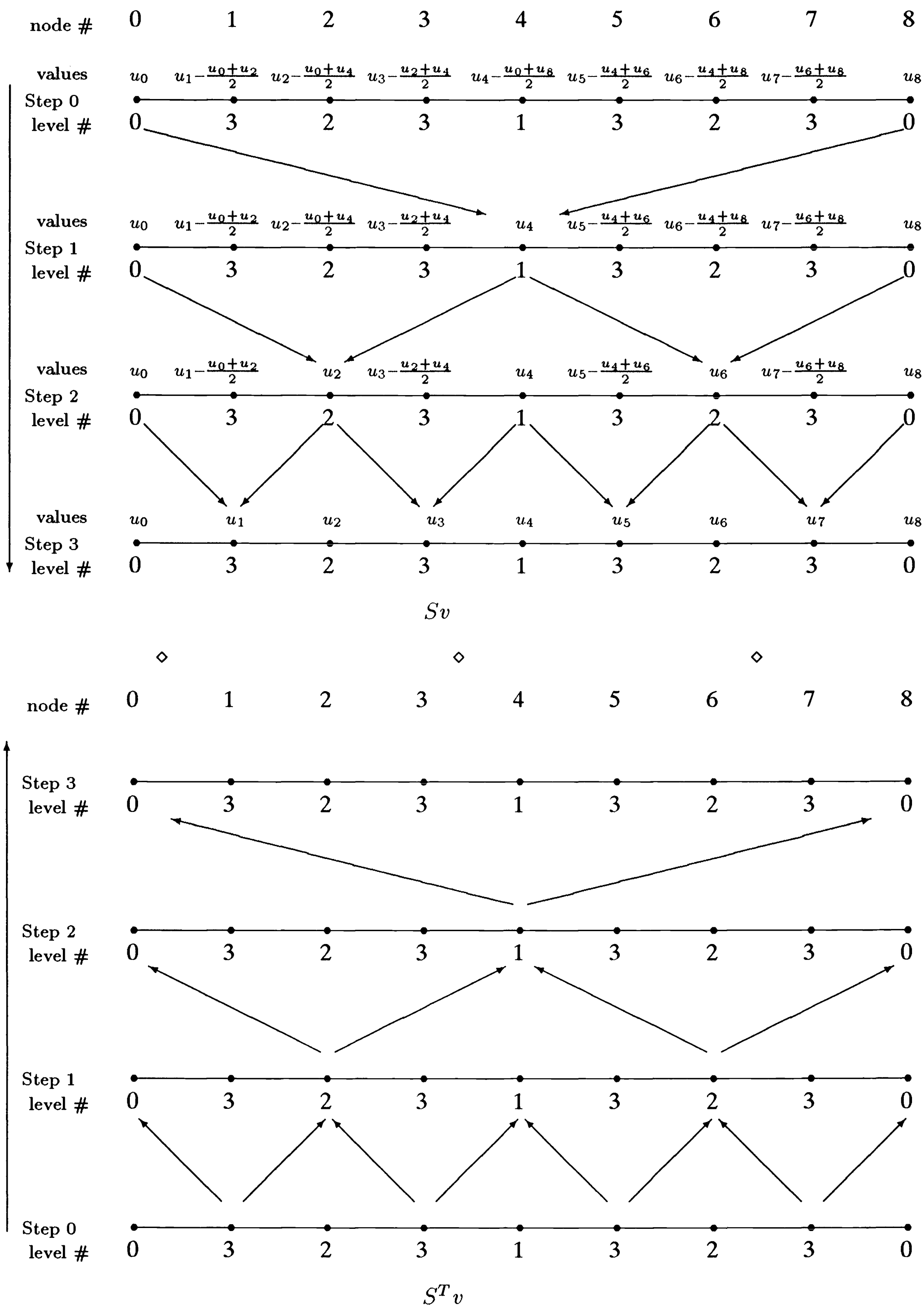
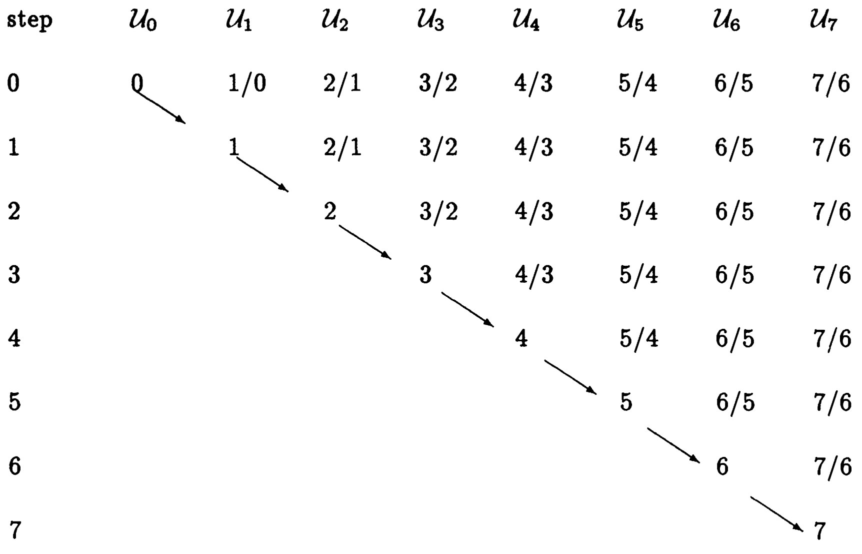

node #
values
Step 0
level #
Step 1
Step 2
Step 3
u_0 $u_1-\frac{u_0+u_2}{2}$ $u_2-\frac{u_0+u_4}{2}$ $u_3-\frac{u_2+u_4}{2}$ $u_4-\frac{u_0+u_8}{2}$ $u_5-\frac{u_4+u_6}{2}$ $u_6-\frac{u_4+u_8}{2}$ $u_7-\frac{u_6+u_8}{2}$ u_8
u_0 u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8
Sv
S^Tv

FIG. 5. *Parallel Algorithm* I *for the* IHB *preconditioner.*



step	$\mathcal{U}_0$	$\mathcal{U}_1$	$\mathcal{U}_2$	$\mathcal{U}_3$	$\mathcal{U}_4$	$\mathcal{U}_5$	$\mathcal{U}_6$	$\mathcal{U}_7$
0	0	1/0	2/1	3/2	4/3	5/4	6/5	7/6
1		1	2/1	3/2	4/3	5/4	6/5	7/6
2			2	3/2	4/3	5/4	6/5	7/6
3				3	4/3	5/4	6/5	7/6
4					4	5/4	6/5	7/6
5						5	6/5	7/6
6							6	7/6
7								7

FIG. 6. *Symbolic representation of Parallel Algorithm* I Sv.

We now present a parallel algorithm for the BPX preconditioner that is based on Parallel Algorithm I for the HB preconditioner and also costs $O(j)$. From (13), we have

$$M^{-1} = h_j^{2-d}I + \hat{S}_j\{h_{j-1}^{2-d}I + \hat{S}_{j-1}[h_{j-2}^{2-d}I + \hat{S}_{j-2}(\cdots)\hat{S}_{j-2}^T]\hat{S}_{j-1}^T\}\hat{S}_j^T. \tag{19}$$

A high-level description of the BPX algorithm based on (19) is as follows.

Algorithm BPX
```
1  $v^j = v$
2  for $k = j$ down to 1, do
3      $v^{k-1} = \hat{S}_k^T v^k$
4  end
5  $w^0 = h_0^{2-d} v^0$
6  for $k = 1$ to $j$, do
7      $w^k = h_k^{2-d} v^k + \hat{S}_k w^{k-1}$
8  end
comment: $M^{-1}v = w^j$
```

This algorithm has been developed independently by Xu and Qin [15]. We now discuss how to implement it efficiently on a parallel computer.

As shown in §2, $\hat{S}_k$ is a block submatrix of S_k. Thus we may use Parallel Algorithm I for the HB preconditioner to compute $\hat{S}_k v$ and $\hat{S}_k^T v$ as follows.

Algorithm $\hat{S}_k v$
```
1  for all $i \in \mathcal{U}_k$
2      $X(i) = (X(I1(i)) + X(I2(i)))/2$
3  all $i$ in parallel
```

Algorithm $\hat{S}_k^T v$
```
1  for all $i \in \mathcal{U}_k$
2      $X(I1(i)) = X(I1(i)) + X(i)/2$
3      $X(I2(i)) = X(I2(i)) + X(i)/2$
4  all $i$ in parallel
```

where $I1$ and $I2$ are defined as in Parallel Algorithm I for the HB preconditioner. We summarize the above algorithms for BPX as follows.

Parallel Algorithm BPX

```
1  for all $i \in \mathcal{N}_j$, do
2       $v^j(i) = v(i)$
3  all $i$ in parallel
4  for $k = j$ down to 1, do
5       for all $i \in \mathcal{U}_k$, do
6            $v^{k-1}(I1(i)) = v^k(I1(i)) + v^k(i)/2$
7            $v^{k-1}(I2(i)) = v^k(I2(i)) + v^k(i)/2$
8       all $i$ in parallel
9  next $k$
10 for all $i \in \mathcal{N}_0$, do
11      $X(i) = h_0^{2-d} v^0(i)$
12 all $i$ in parallel
13 for $k = 1$ to $j$, do
14      for all $i \in \mathcal{U}_k$
15           $X(i) = (X(I1(i)) + X(I2(i)))/2$
16      all $i$ in parallel
17      for all $i \in \mathcal{N}_k$, do
18           $X(i) = h_k^{2-d} v^k + X(i)$
19      all $i$ in parallel
20 next $k$
```

For two-dimensional problems, the scaling factors $\{h_k^{2-d}\}_{k=0}^{j}$ equal 1 and this algorithm costs exactly the same as Parallel Algorithm I for HB. For three or higher dimensional problems, the BPX preconditioner needs $j+1$ parallel multiplications to evaluate the action of the scaling by the factors $\{h_k^{2-d}\}_{k=0}^{j}$. The HB preconditioner also needs to be scaled by a diagonal matrix D_d consisting of $\{h_k^{2-d}\}_{k=0}^{j}$. But the action of D_d can be evaluated with only one parallel multiplication. Therefore for three or higher dimensional problems, the parallel BPX algorithm requires j more multiplications than Parallel Algorithm I for HB.

Note that both Figs. 5 and 6 indicate that Parallel Algorithm I for the HB preconditioner leaves many processors idle, especially in the part of the computation corresponding to the coarse grids. We now describe an alternative parallel algorithm that makes use of some of these idle processors and requires fewer steps than parallel Algorithm I. The new algorithm is depicted symbolically in Figs. 7 and 8, which are analogous to Figs. 5 and 6. The computation of Sv can be defined precisely by the following three rules.

RULE I. Let $l = \sum_{s=0}^{\infty} c_s 2^s$ (c_s =0 or 1). $\mathcal{U}_l$ is updated at step $k+1$ if and only if $c_k = 1$.

This rule determines when a level is updated. For example, $7 = 2^0 + 2^1 + 2^2$ so that $\mathcal{U}_7$ is updated at steps 1, 2, and 3, whereas $5 = 2^0 + 2^2$, so that $\mathcal{U}_5$ is updated at steps 1 and 3, but not at step 2. (See Fig. 8.)

To specify the second rule, we will say that level l depends on level d if all processors handling $i \in \mathcal{U}_l$ contain $(\mathcal{I}_l u - \mathcal{I}_d u)(x)$ (where $\mathcal{I}_d \equiv 0$ for $d < 0$).

RULE II. Let $l = \sum_{k=0}^{\infty} c_k 2^k$ (c_k =0 or 1). Level l depends on level $d = l - 2^k + \sum_{s=0}^{k-1}(1 - c_s)2^s$ after step k.

To see how this rule applies, consider the case of $l = 5 = 2^0 + 2^2$. (See Fig. 8.) Initially, $d = 5 - 2^0$, so that level 5 depends on level 4. After step 1, $d = 5 - 2^1$ so that

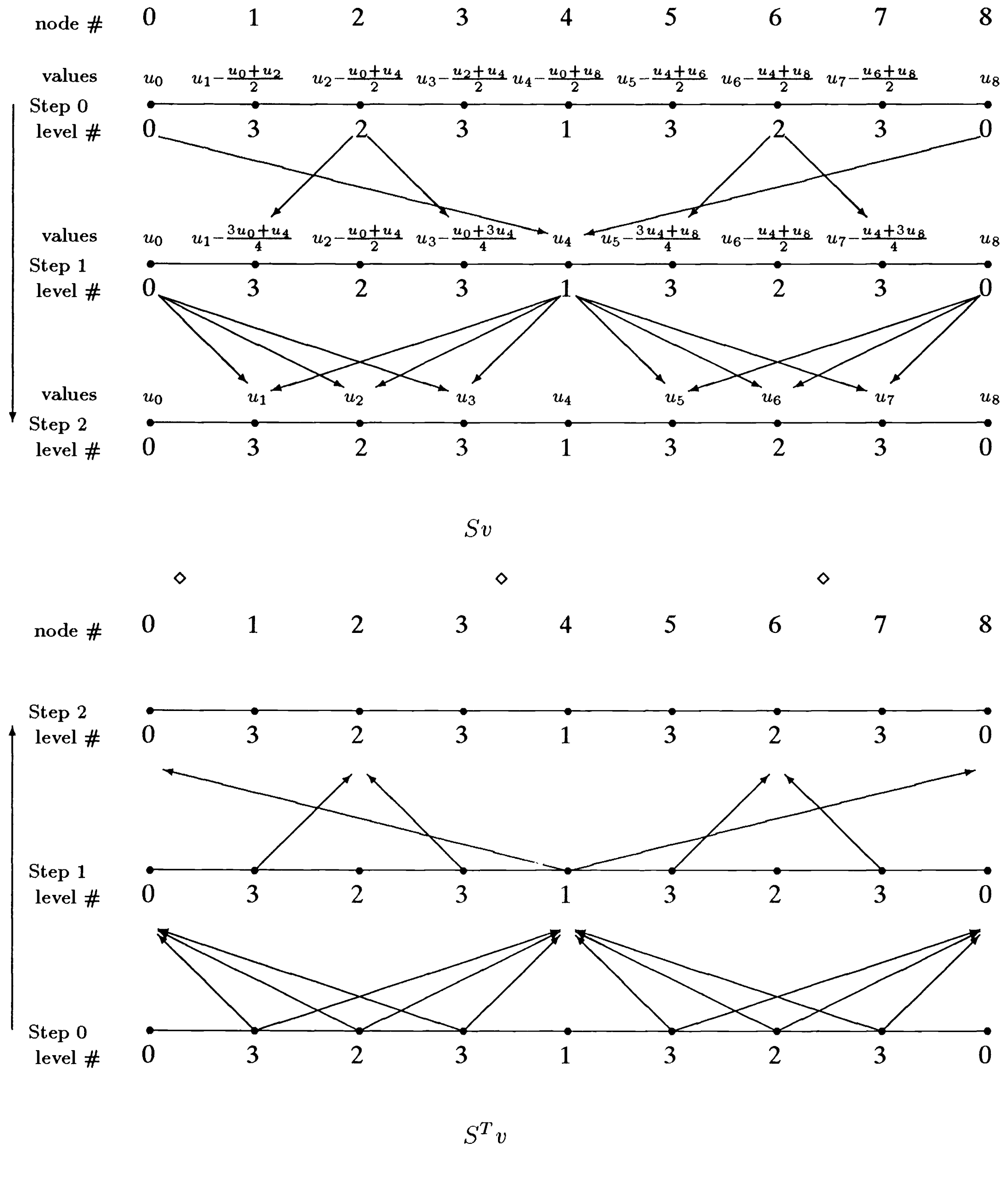


FIG. 7. *Parallel Algorithm* II *for the* IHB *preconditioner.*

level 5 depends on level 3, and after step 2, $d = 5-2^2+\sum_{s=0}^{1}(1-c_s)2^s = 5-4+(0+2)$, so that level 5 still depends on level 3. Note that according to Rule I, the update of level 5 with level 3 is not performed until step 3.

To specify the third rule, we need some additional notation. Let $J = \lceil \log_2 j \rceil$. Given a node i, let $l(i)$ denote its level number, i.e., $l(i)$ is the integer such that $i \in \mathcal{U}_{l(i)}$.

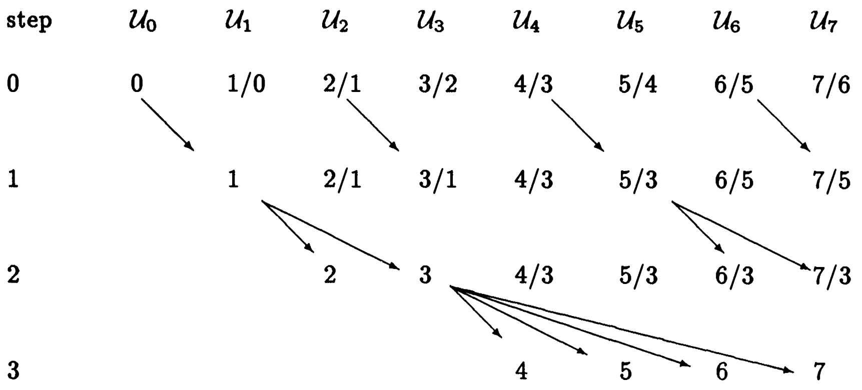

step
$\mathcal{U}_0$ $\mathcal{U}_1$ $\mathcal{U}_2$ $\mathcal{U}_3$ $\mathcal{U}_4$ $\mathcal{U}_5$ $\mathcal{U}_6$ $\mathcal{U}_7$
0: 0 1/0 2/1 3/2 4/3 5/4 6/5 7/6
1: 1 2/1 3/1 4/3 5/3 6/5 7/5
2: 2 3 4/3 5/3 6/3 7/3
3: 4 5 6 7

FIG. 8. *Symbolic representation of Parallel Algorithm* II Sv.

Let $(c_0(i),\dots,c_{J-1}(i))$ be the vector such that $l(i)=\sum_{s=0}^{J-1}c_s(i)2^s$ ($c_s=0$ or 1), let $(d_0(i),\dots,d_J(i))$ be the vector such that level $l(i)$ depends on level $d_k(i)$ after step k in the sense of Rule II. Note that $\{l(i)\}$, $\{c_k(i)\}$ and $\{d_k(i)\}$ are just more precise delineations of the quantities l, c_k and d used in Rules I and II. Finally, let $I1_k(i)$ (or $I2_k(i)$) denote the nodes closest to node i among all nodes from $\mathcal{N}_{d_k(i)}$ on the right (or left) side of i. Thus, i lies between $I1_k(i)$ and $I2_k(i)$ and no node from $\mathcal{N}_{d_k(i)}$ lies between $I1_k(i)$ and $I2_k(i)$. We call $I1_k(i)$, $I2_k(i)$ the parents at level k of node i. For example, after step 1 of the computation of Sv, the parents of node 7 are $I1_1(7)=8$ and $I2_1(7)=4$. These are the sources of the arrows between steps 1 and 2 in the top part of Fig. 7. Let X be the vector initialized with the hierarchical basis coefficients and let $\mathcal{N}_k=\emptyset$ for $k<0$.

RULE III. When level l is to be updated according to Rule I, the data at a node $i\in\mathcal{U}_k$ will be updated in the following two steps:

```
1   if $I1_{k-1}(i)\in\mathcal{N}_{d_{k-1}(i)}\setminus\mathcal{N}_{d_k(i)}$, then
2       $X(i)=X(i)+\frac{I2_{k-1}(i)-i}{I2_{k-1}(i)-I1_{k-1}(i)}X(I1_{k-1}(i))$
3   end if
4   if $I2_{k-1}(i)\in\mathcal{N}_{d_{k-1}(i)}\setminus\mathcal{N}_{d_k(i)}$, then
5       $X(i)=X(i)+\frac{I1_{k-1}(i)-i}{I1_{k-1}(i)-I2_{k-1}(i)}X(I2_{k-1}(i))$
6   end if
```

Lines 1 and 4 of Rule III are necessary for Rule II to be satisfied, i.e., for all processors handling $x_i\in\mathcal{U}_l$ to contain $(\mathcal{I}_lu-\mathcal{I}_{d_k(i)}u)(x_i)$. Note that given the values $X(I1_{k-1}(i))$ and $X(I2_{k-1}(i))$ at the parents $I1_{k-1}(i)$ and $I2_{k-1}(i)$, the value of their linear interpolant at node i is

$$\left[\frac{I2_{k-1}(i)-i}{I2_{k-1}(i)-I1_{k-1}(i)}\right]X(I1_{k-1}(i))+\left[\frac{I1_{k-1}(i)-i}{I1_{k-1}(i)-I2_{k-1}(i)}\right]X(I2_{k-1}(i)),$$

where the terms in brackets are the corresponding linear interpolation coefficients.

To see how Rule III applies, consider node $i=1$ in the example shown by Fig. 7. (All nodes are numbered from left to right starting with 0.) Since $l(1)=3=2^0+2^1$, node 1 will be updated at steps 1 and 2 according to Rule I, and node 1 will depend on levels 2, 1 and -1 after step 0, 1 and 2, respectively, according to Rule II, i.e., $(d_0(i),d_1(i),d_2(i))$=(2,1,-1). Here $(I1_0(i),I1_1(i))=(2,4)$ and $(I2_0(i),I2_1(i))=(0,0)$. Thus, one adds only $\frac{1}{2}X(2)$ to $X(1)$ at step 1, since $I1_0(i)=2\in\mathcal{N}_2\setminus\mathcal{N}_1$ and $I2_0(i)=$

$0 \notin \mathcal{N}_2 \setminus \mathcal{N}_1$. At step 2, one adds both $\frac{1}{4}X(4)$ and $\frac{3}{4}X(0)$ to $X(1)$, since both $I1_1(i) = 4$ and $I2_1(i) = 0$ are in $\mathcal{N}_1 \setminus \mathcal{N}_{-1} = \mathcal{N}_1$.

Combining the three rules, we formulate the new parallel algorithm for computing Sv as follows.

Parallel Algorithm II Sv

```
1  for $k = 1$ to $J$, do
2    for all $i \in \mathcal{U}_{l(i)}$ such that $s_{k-1}(i) = 1$, do
3      If $I1_{k-1}(i) \in \mathcal{N}_{d_{k-1}(i)} \setminus \mathcal{N}_{d_k(i)}$, then
4        $X(i) = X(i) + \frac{I2_{k-1}(i)-i}{I2_{k-1}(i)-I1_{k-1}(i)} X(I1_{k-1}(i))$
5      end if
6      If $I2_{k-1}(i) \in \mathcal{N}_{d_{k-1}(i)} \setminus \mathcal{N}_{d_k(i)}$, then
7        $X(i) = X(i) + \frac{I1_{k-1}(i)-i}{I1_{k-1}(i)-I2_{k-1}(i)} X(I2_{k-1}(i))$
8      end if
9    all $i$ in parallel
10 next $k$
```

The analogous algorithm for computing $S^T v$ is as follows:

Parallel Algorithm II $S^T v$

```
1  for $k = J$ down to 1, do
2    for all $i \in \mathcal{U}_{l(i)}$ such that $s_{k-1}(i) = 1$, do
3      if $I1_{k-1}(i) \in \mathcal{N}_{d_{k-1}(i)} \setminus \mathcal{N}_{d_k(i)}$, then
4        $X(I1_{k-1}(i)) = X(I1_{k-1}(i)) + \frac{I2_{k-1}(i)-i}{I2_{k-1}(i)-I1_{k-1}(i)} X(i)$
5      end if
6      if $I2_{k-1}(i) \in \mathcal{N}_{d_{k-1}(i)} \setminus \mathcal{N}_{d_k(i)}$, then
7        $X(I2_{k-1}(i)) = X(I2_{k-1}(i)) + \frac{I1_{k-1}(i)-i}{I1_{k-1}(i)-I2_{k-1}(i)} X(i)$
8      end if
9    all $i$ in parallel
10 next $k$
```

For higher dimensional problems, Rule III has to be modified slightly. For example, the modification for the two-dimensional problem is as follows: (a) For each node $i \in \mathcal{U}_l$, there are three parents $I1_k(i), I2_k(i), I3_k(i)$ at level $k < l$, which are defined as the three vertices of the element at level k including node i inside or on the boundary. (For the case on the boundary, there are two elements at level k including i, and we choose one of them.) (b) The linear interpolation coefficients are the barycentric coordinates of nodes i with respect to $I1_k(i), I2_k(i), I3_k(i)$. The barycentric coordinates $\lambda_j = \lambda_j(x), 1 \le j \le d+1$ of any point $x \in \mathbf{R}^d$, with respect to the $(d+1)$ vertices a_j of a simplex, are the (unique) solutions of the linear system:

$$\begin{cases} \sum_{j=1}^{d+1} a_j \lambda_j = x \\ \sum_{j=1}^{d+1} \lambda_j = 1. \end{cases}$$

Now consider the costs of Parallel Algorithm II. Rule I implies that for all nodes at level l, no update is required after step $\lceil \log_2 l \rceil$. Consequently, this algorithm requires at most $\lceil \log_2 j \rceil$ parallel steps. A direct comparison of its costs with those of Algorithm I for the HB preconditioner is somewhat problematic. For the computation of Sv, each step of Algorithm II requires $O(1)$ arithmetic operations, so that the total cost of arithmetic is $O(\lceil \log_2 j \rceil)$. However, for step k of the computation of $S^T v$, as many as $O(4^{\lceil log_2 j \rceil - k})$ processors send data to one individual processor, say processor i, and the numerical sum of these data is then stored. (See Fig. 7.) If processor i performs the summation serially, then this leads to an arithmetic cost of $C \sum_{k=1}^{\lceil \log_2 j \rceil} 4^{\lceil \log_2 j \rceil - k} = O(j)$. This is of the same order of magnitude as the cost of Algorithm I. Moreover, it is evident that Algorithm II has more complex communication patterns than Algorithm I. To get some further insight into the comparative costs of these algorithms, consider their performance on the Connection Machine. This computer has the feature that if a set of numbers is sent to an individual processor and the numerical sum is to be stored there, then for items that meet at other processors en route to their final destination, partial sums are computed. This process of "enroute combining" reduces congestion and speeds up both communication and arithmetic, because as messages are combined their number is reduced. For one-dimensional problems, Fig. 9 shows the CPU times of the two algorithms. The results indicate that for both the Sv and $S^T v$ computations, the cost of Algorithm II is close to $O(\lceil \log_2 j \rceil)$. Moreover, the new algorithm is more efficient than Algorithm I for all but the smallest problems. In two or higher dimensions, a larger number of levels is required for Algorithm II to be superior to Algorithm I. Based on an extrapolation of experimental results for two-dimensional problems up to eight levels (the largest number of levels we could fit into memory), we estimate that Algorithm I becomes more efficient for roughly 20 or more levels.

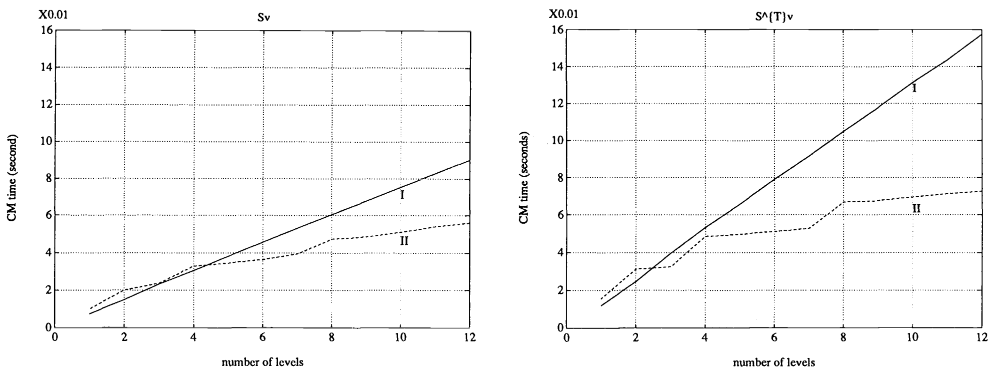


FIG. 9. *Comparison between two algorithms for the* HB *preconditioner in one dimension, with bit serial double precision and* 8K *CM.*

REFERENCES

[1] L. M. ADAMS AND M. E. G. ONG, *A comparison of preconditioners for* GMRES, *Parallel Computations and their Impact on Mechanics*, A. K. Noor, ed., AMD-Vol. 86, 1987.

[2] R. E. BANK, T. F. DUPONT, AND H. YSERENTANT, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[3] J. H. BRAMBLE, J. E. PASCIAK, AND J. XU, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[4] T. F. CHAN, C.-C. J. KUO, AND C. H. TONG, *Parallel elliptic preconditioners*: *Fourier analysis and performance on the connection machine*, Comput. Phys. Comm., 53 (1989), pp. 237–252.

[5] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

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

[7] A. Greenbaum, C. Li, and H. Z. Chao, *Comparison of linear system solvers applied to diffusion-type finite element equations*, Numer. Math., 56 (1989), pp. 529–546.

[8] J. A. Meijerink and H.A. van der Vorst, *An iterative solution method for linear equation systems of which the coefficients matrix is a symmetric* M-*matrix*, Math. Comp., 31 (1977), pp. 148–162.

[9] M. E. G. Ong, *The 3D linear hierarchical basis preconditioner and its shared memory parallel implementation*, in Proc. Second Internat. Conf. on Vector and Parallel Comput. Issues in Appl. Res. and Development, Tromso, Norway, June 6–10, 1988.

[10] ———, *Hierarchical Basis Preconditioners for Second Order Elliptic Problems in Three Dimensions*, Ph.D. thesis, Univ. of Washington, Seattle, 1989.

[11] P. Oswald, *On discrete norm estimates related to multilevel preconditioners in the finite element method*, in Proc. Internat. Conf. Theory of Functions, Varna, 1991, to appear.

[12] C. H. Tong, *Parallel Preconditioned Conjugate Gradient Methods for Elliptic Partial Differential Equations*, Ph.D. thesis, Univ. of California, Los Angeles, 1990.

[13] J. Xu, *Theory of Multilevel Methods*, Tech. Rep. AM 48, Department of Mathematics, The Pennsylvania State Univ., University Park, PA, July 1989.

[14] ———, *Iterative Methods by Space Decomposition and Subspace Correction: A Unifying Approach*, Tech. Rep. AM 67, Department of Mathematics, The Pennsylvania State Univ., University Park, PA, 1990; SIAM Rev., 34 (1992), pp. 581–613.

[15] J. Xu and J. Qin, *On Some Multilevel Preconditioners*, Tech. Rep. AM 90, Department of Mathematics, The Pennsylvania State Univ., University Park, PA, 1989.

[16] X. Ye, *Parallel Implementation of Multilevel Algorithm On the Hypercube Multiprocessor*, Master's thesis, The Pennsylvania State Univ., University Park, PA, March 1990.

[17] H. Yserentant, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.

[18] ———, *Two preconditioners based on the multi-level splitting of finite element spaces*, Numer. Math., 58 (1990), pp. 163–184.

[19] X. Zhang, *Studies in Domain Decomposition: Multi-level Methods and the Biharmonic Dirichlet Problem*, Ph.D. thesis, New York University, New York, 1991.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 914-935, July 1993

PARALLEL COMPACT FFTs FOR REAL SEQUENCES*

RICHARD B. PELZ†

Abstract. Eight algorithms for the in-place, compact, fast Fourier transform (FFT) of real and conjugate-symmetric sequences are presented for single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) distributed-memory multiprocessors. The "conditional ordering" is introduced for conjugate-symmetric sequences as the natural ordering that leads to minimal communication costs. For a transform of a real, naturally ordered, input sequence of length N distributed evenly across P processors, $2+\log_2 P$ nearest neighbor communication exchanges of $\frac{1}{2}N/P$ contiguous elements (i-cycles) are necessary. For a transform of a conjugate-symmetric, conditionally ordered input sequence, $1+\log_2 P$ i-cycles are necessary. In all cases the computational complexity is $\frac{5}{2}N/P\log_2 N$, and there are no extra memory requirements. The algorithms require about half the communication of a parallel FFT of a real sequence that involves a complex FFT and pre- or postprocessing. Communication complexity is based on the hypercube topology; the algorithms, however, can be implemented on general connection topologies. Some of the schemes are implemented on an nCUBE/2 MIMD multiprocessor, and timings are given.

Key words. fast Fourier transform, parallel algorithms

AMS subject classifications. 65J20, 65Y05, 65Y20

1. Introduction.

Cooley, Lewis, and Welch (CLW) [4] showed that the discrete Fourier transform of a real sequence can be accomplished with a Fourier transform of an associated complex sequence of half the length and a pre- or postprocessing step. Coupled with the FFT of complex data, the CLW algorithm provides an efficient way to compute the real Fourier transform on scalar and vector computers. A similar algorithm for two real sequences uses one FFT of an associated complex vector and pre- or postprocessing.

In Table 1, we give information about the four common, in-place, FFT algorithms for a complex sequence. The first two, CT1 and CT2, were developed by Cooley and Tukey [5]. The latter two, GS1 and GS2, were developed by Gentleman and Sande [7]. Table 2 contains the eight possible FFTs for a real sequence.

TABLE 1
Characteristics of in-place FFTs. RU *refers to roots of unity.*

Name	Input order	Output order	RU order	Formal inverse
CT1	natural	bit-reversed	bit-reversed	GS2
CT2	bit-reversed	natural	natural	GS1
GS1	natural	bit-reversed	natural	CT2
GS2	bit-reversed	natural	bit-reversed	CT1

Salmon [11] was the first to implement the complex FFT on a parallel machine. A refined in-place parallel algorithm, which was given independently by Swarztrauber [13] and Walker [14], has the minimum amount of interprocessor communication on hypercube, distributed-memory multiprocessors: $\log_2 P+1$ nearest-neighbor communication exchanges of $N/(2P)$ complex numbers.

* Received by the editors April 16, 1991; accepted for publication (in revised form) June 10, 1992. This work was supported by National Science Foundation grant EET 88-08780 and by Office of Naval Research grant N00014-89-J1320.

† Mechanical and Aerospace Engineering, Rutgers University, Piscataway, New Jersey 08855-0909.

TABLE 2
Characteristics of in-place FFTs *for real and conjugate symmetric sequences.* RU *refers to roots of unity,* CS *refers to conjugate symmetric.*

Name	Input order	Output order	RU order	Formal inverse
CT1R	real natural	cs bit-reversed	bit-reversed	GS2CS
CT1CS	cs natural	real bit-reversed	bit-reversed	GS2R
CT2R	real bit-reversed	cs natural	natural	GS1CS
CT2CS	cs bit-reversed	real natural	natural	GS1R
GS1R	real natural	cs bit-reversed	natural	CT2CS
GS1CS	cs natural	real bit-reversed	natural	CT2R
GS2R	real bit-reversed	cs natural	bit-reversed	CT1CS
GS2CS	cs bit-reversed	real natural	bit-reversed	CT1R

On the distributed-memory multiprocessor, the Swarztrauber–Walker FFT can be used as the first step of the CLW method for the transform of a real sequence; however, we shall see that the postprocessing step requires as much communication as the complex FFT on hypercubes. Furthermore, it can be shown that for input data with n symmetries (e.g., one symmetry is for real transforms, two are for cosine and sine, three are for quarter-even and quarter-odd), the CLW method requires n times the amount of communication of the associated complex FFT.

On many distributed-memory multiprocessors, communication time is already a significant part of the total time in the FFT. Increasing the communication time lowers the parallel efficiency and reduces the performance of numerical methods that have the real FFT as an integral part.

A class of FFTs for symmetric sequences, called compact algorithms, were developed by Swarztrauber [12], Bergland [1], and Briggs [2]. They showed that, by using the symmetries during each stage, FFTs of symmetric data could be generated with similar complexity to the CLW FFT, but without the pre- or postprocessing. Bergland attributes this idea to Edson.

Swarztrauber [12] described the compact FFT, which is equivalent to GS2R or CT2R depending on how the roots of unity are computed. The real input is in bit-reversed order and conjugate-symmetric output is in natural order. This transform is known as the Edson transform. The inverse transforms are CT1CS and GS1CS. Briggs [2] developed the compact FFT, which is an extension of GS1R or CT1R, again depending on the roots of unity. The real input is in natural order and the conjugate-symmetric output is in bit-reversed order. The inverse transforms are CT2CS and GS2CS.

This paper presents algorithms for the eight compact FFTs of real data given in Table 2, which can be implemented efficiently on distributed-memory multiprocessors [17]. Similar work is also reported in [15] and [16] in the context of global climate models. All the schemes were designed according to the following principles.

1. All communication is in the form of i-cycles, nearest-neighbor exchanges on a hypercube. The communication is kept to a minimum.
2. The instructions are similar on each processor. This will allow for concurrency on both MIMD and SIMD multiprocessors.
3. Real and imaginary parts of each element are contiguous in memory and not distributed across processors.
4. Computational complexity is similar to that of the sequential schemes (divided by the number of processors).
5. The algorithms require no extra memory.

For an input sequence that is conjugate symmetric, a new ordering which we shall call "conditional ordering" replaces the natural ordering. Using natural ordering of a conjugate-symmetric sequence resulted in an awkward and inefficient communication structure. Conditional ordering is similar to natural ordering, but results in lower communication complexity.

As in the Swarztrauber–Walker FFT for a complex sequence, the output is in "processor-transposed," bit-reversed order. To reorder the output requires at least as much communication time as the FFT. We shall not consider this reordering nor shall we consider the conversion of a conjugate-symmetric sequence from conditional to natural ordering. Because of this, these algorithms may not be useful for general software libraries. Many applications in computational physics, such as evaluation of convolutions and nonlinear terms (e.g., the Fourier pseudospectral method for fluid dynamics [3]), do not require a special ordering of the output sequence. The algorithms presented here are aimed at such applications.

The structure of the paper is as follows. Information on communication, conditional ordering, and reduced operations due to symmetries are described in § 2. In § 3 the serial and parallel versions of the CLW FFT algorithm are presented. In §§ 4 and 5, the Briggs and Edson transforms, respectively, are reviewed, the parallel algorithms are developed, and performance analyses are given. Numerical results and timings are presented in § 6.

2. Preliminaries. In this section we review the *i*-cycle permutation that will be used for communication, define the conditional ordering, and define the reduced operations that occur due to the symmetries of the input sequence.

The length of the real sequence is N, and the number of processors is P. We denote $m = \log_2 N$ and $d = \log_2 P$, the dimension of the hypercube. The input sequence is distributed evenly and contiguously between P processors. We also desire an even distribution of the output sequence. On a hypercube connection topology, the subsequences are sequentially mapped onto the processors. The index k of the input sequence is then written in binary form as

$$k_{m-1}\, k_{m-2} \dots k_{m-d} \,|\, k_{m-d-1} \dots k_0. \tag{2.1}$$

The binary index of the processor label is to the left of the vertical bar, and the binary index of the local memory is to the right.

We shall use an *i*-cycle [13], which is a binary index-digit permutation, to describe the communication exchanges in the compact FFTs. Swarztrauber [13] showed that by using *i*-cycles in his complex FFT, the communication is minimized. In an *i*-cycle, a digit in the processor label (called the target digit) and the left-most digit in the local memory (called the pivot) are switched. This corresponds to an exchange of the last half of the subsequence in the processor whose target digit is zero with the first half of the subsequence in the processor whose target digit is one. It is a nearest-neighbor exchange of $N/2P$ contiguous elements. No extra memory is required.

Definition. Conditional ordering of a conjugate-symmetric sequence is a way of packing a conjugate-symmetric sequence of length N into a complex array of length $N/2$. First, we select from the conjugate-symmetric sequence only those elements whose binary index has the following property: the binary place to the left of the first nonzero bit from the right is zero. For example, for $N = 16$, the sequence is

$$\{0, 1, 2, 4, 5, 8, 9, 10, 13\}. \tag{2.2}$$

This reduces the set of elements to $N/2 + 1$. Since mode 0 and $N/2$ are real, we pack them into a complex mode 0. The elements are then placed into position within the set equal to each element's index without the "zero" place. For example,

$$\begin{aligned}&\{(0,8)\,1,2,5,4,9,10,13\} \quad \text{for } N=16 \quad \text{and}\\&\{(0,16),1,2,5,4,9,10,13,8,17,18,21,20,25,26,29\} \quad \text{for } N=32.\end{aligned} \tag{2.3}$$

Due to the way in which we take advantage of the symmetries of the sequences, this ordering is necessary to streamline the communication. I-cycle communication and nonnearest-neighbor communication results if this ordering is used instead of the first $N/2+1$ elements of a naturally ordered conjugate-symmetric array.

DEFINITION. Second conditional ordering of a conjugate-symmetric sequence is another way of packing. We first take a sequence of integers $0, 1, \ldots, \mathrm{N}-1$; we then select $N/2+1$ elements of the sequence that have a zero bit to the right of the first nonzero bit from the left. We pack the first two elements together into a complex integer. The reduced sequence for $N=32$ is

$$\{(0,1)\;2\;4\;5\;8\;9\;10\;11\;16\;17\;18\;19\;20\;21\;22\;23\}. \tag{2.4}$$

The bit-reversed indexing of this sequence is

$$\{(0,16)\;8\;4\;20\;2\;18\;10\;26\;1\;17\;9\;25\;5\;21\;13\;29\}. \tag{2.5}$$

Two reduced operations that result from the symmetries of the input sequence are now presented.

DEFINITION. Operation A. Given two real numbers x and y, we define operation A as $x+y$ replaces x and $x-y$ replaces y.

DEFINITION. Operation B. If we let the output number z be complex, then operation B is defined as x is the real part of z and y is the imaginary part of z. z then replaces (x, y) in memory.

3. The parallel CLW algorithm.

In this section we present the parallel method for the CLW algorithm. We begin by reviewing the serial algorithm.

3.1. The serial CLW FFT.

Let the Fourier transform of a sequence of N real numbers, $Y(j), j=0,1,\ldots,N-1$, be $C(n)$, $n=0,1,\ldots,N-1$. C is conjugate symmetric, $C(n)=C(N-n)^*$, and the imaginary parts of $C(0)$ and $C(N/2)$ are zero.

The first step of the CLW algorithm is to create the complex array X from Y, as follows:

$$X(j)=Y(2j)+iY(2j+1), \qquad j=0,1,\ldots,N/2-1, \tag{3.1}$$

where $i=\sqrt{-1}$. Then an $N/2$-point complex FFT, $X\leftrightarrow A$, is performed. The final step is to perform the following operation:

$$\begin{aligned}C(n)&=\tfrac{1}{4}\{A(n)+A^*(N/2-n)-i\omega_{N^n}^*[A(n)-A^*(N/2-n)]\},\\C(N/2-n)&=\tfrac{1}{4}\{A(N/2-n)+A^*(n)+i\omega_{N^n}[A(N/2-n)-A^*(n)]\},\end{aligned} \tag{3.2}$$

where $n=0,1,\ldots,N/4$, ω_{N^n} is the nth root of N roots of unity and $(\cdot)^*$ denotes the complex conjugate.

3.2. The parallel CLW FFT.

For the parallel algorithm, we assume that Y is evenly distributed across P processors. If $P\le N/2$, then (3.1) is done within each processor and no communication is required. The ordering of the input sequence is such that X is bit-reversed. The complex FFT is performed using the inplace algorithm of Swarztrauber [13] and Walker [14]. The transformed sequence is in sequential order. In addition, the processor binary index is transposed such that $(n_0\, n_{d-1}\, n_{d-2}\ldots n_1)$.

Operation (3.2) requires a pair of elements reflected about the midpoint of the sequence and hence requires communication to complete. If A is sequentially ordered, and $P = N/2$, then the odd index pairs (n and $N/2 - n$) are on opposite vertices of a $d-1$ hypercube. In this case the Hamming distance is $d-1$. If $P < N/2$, then the distance increases to d, because the right-most binary place of the index is not in the processor label. The processor transposition has no effect on the Hamming distance.

Operation (3.2) requires only one major communication of data if the binary reflected grey code mapping is used. Unfortunately, this also increases the communication distance in the FFT to next-nearest-neighbors. Both mappings produce a similar product of distance times amount of communicated data.

We are led to the conclusion that operation (3.2), while requiring little computation, requires nearly as much communication (defined as message length times Hamming distance) as the complex FFT. For an example of the degradation of performance in the Fourier pseudospectral method for Navier–Stokes simulations, see [8]. For a discussion of this algorithm on distributed SIMD and MIMD multiprocessors, see [9] and [10] and [16], respectively.

4. The parallel, compact FFT for a real, naturally ordered input sequence.

In this section, we present an extension of the compact FFT presented by Briggs [2] for the transform of a real, naturally ordered input sequence (CT1R). His approach was to develop an FFT for a conjugate-symmetric sequence in bit-reversed order in a manner similar to Swarztrauber [12]. The output sequence is real and in natural order. The inverse of this transform is then what is desired.

4.1. A serial CT1R algorithm.

There are many ways to implement the CT1R transform. We shall describe one that is easily parallelizable. We denote the binary index of the real input sequence as

$$(k_{m-1}\, k_{m-2} \ldots k_0), \tag{4.1}$$

and the binary index of the complex output sequence as

$$(n_0\, n_1 \ldots n_{m-1}). \tag{4.2}$$

Because the output is complex, only $m-1$ bits have the range $[0, 1]$. We shall pack the zero- and $N/2$-mode coefficients that are real into a zero-mode complex number.

The fact that the input array is real can be used during each stage of the transform. The first stage, the summation over k_{m-1}, has real output, if the input is real. This is operation A as defined in § 2. The binary index k_{m-1} of the input vector is replaced by n_{m-1} in the output index.

The second stage proceeds in two parts. If $n_{m-1} = 0$, then the summation over k_{m-2} is similar to that of the first stage. If $n_{m-1} = 1$, then the output of this stage is a complex number in which the $k_{m-2} = 0$ element is the real part and the associated $k_{m-2} = 1$ element is the imaginary part. The element with $n_{m-2} = 1$ is the complex conjugate of the element with $n_{m-2} = 0$. Only the $n_{m-2} = 0$ element is calculated. This is operation B as defined in § 2. The real and imaginary parts are placed in contiguous memory. The binary index of the output of the second stage is then

$$\begin{aligned} &(n_{m-1} = 0\; n_{m-2}\, k_{m-3}\, k_{m-4} \ldots k_1\, k_0) \quad \text{and} \\ &(n_{m-1} = 1\; n_{m-2} = 0\; k_{m-3}\, k_{m-4} \ldots k_1\, k_0\, c), \end{aligned} \tag{4.3}$$

where c represents complex elements.

The third stage has three parts. For $n_{m-1} = n_{m-2} = 0$, the summation over k_{m-3} has real input and real output (operation A). For $n_{m-1} = 0$ and $n_{m-2} = 1$, the complex elements are formed as in the preceding stage (operation B). For $n_{m-1}=1$, CT1 operations for a complex input sequence are performed. The binary index of the output is

$$(4.4)\qquad \begin{aligned}&(n_{m-1}=0\ n_{m-2}=0\ n_{m-3}\ k_{m-4}\dots k_1\ k_0),\\ &(n_{m-1}=0\ n_{m-2}=1\ n_{m-3}=0\ k_{m-4}\dots k_1\ k_0\ c),\quad\text{and}\\ &(n_{m-1}=1\ n_{m-2}=0\ n_{m-3}\ k_{m-4}\dots k_1\ k_0\ c).\end{aligned}$$

The remaining stages are similar to the last. After the mth stage, there will be two real numbers with index $n_{m-1}=n_{m-2}=\cdots=n_1=0$; these correspond to the elements of the zero mode and the $N/2$ mode and will be contiguous in memory. The other $N/2-1$ elements are complex and are indexed such that at least one of the m binary digits is one.

The output sequence is in the second-conditional, bit-reversed order. Table 3 illustrates how this ordering occurs with an example.

TABLE 3
Example of the reduced sequence for a compact transform $n=16$. y = yes, the element is needed. n = no, the element is the conjugate of an existing element and is not needed.

Base 10	Base 2	Stage 1	Stage 2	Stage 3	Stage 4	Remaining	Bit-reversed base 10
0	0000	y	y	y	y	0000	0
1	0001	y	y	y	y	0001	8
2	0010	y	y	y	y	0010	4
3	0011	y	y	y	n		
4	0100	y	y	y	y	0100	2
5	0101	y	y	y	y	0101	10 = 6*
6	0110	y	y	n	n		
7	0111	y	y	n	n		
8	1000	y	y	y	y	1000	1
9	1001	y	y	y	y	1001	9 = 7*
10	1010	y	y	y	y	1010	5
11	1011	y	y	y	y	1011	13 = 3*
12	1100	y	n	n	n		
13	1101	y	n	n	n		
14	1110	y	n	n	n		
15	1111	y	n	n	n		

The inverse transform can be accomplished by reversing the stages and operations. The input data should be in the same order as the output of the forward transform. This corresponds to the GS2CS algorithm.

4.2. The parallel CT1R algorithm. Now we present the parallel extension of the CT1R algorithm just desribed. Table 4 shows the binary index of the vector during each step of the transform for $m=6$, $d=3$.

In order to perform the first stage, operation A, the i-cycle with the $(m-1)$st digit as the target is executed. The first stage is then performed on all processors as described above, and the input digit k_{m-1} changes to the output digit n_{m-1}. Anticipating the fact that complex numbers will be created during the second stage, and that real and imaginary parts are to be contiguous, a local rearrangement of the sequence is done. The transpose leaves the index of the array as shown in Table 4. Note that since k_{m-d-1} has moved into the processor label, it must be moved back using an i-cycle to complete the FFT; hence, the number of communications is at least $d+1$.

TABLE 4

Binary index table of the compact transform with a real, naturally ordered input sequence and conjugate-symmetric, bit-reversed order output sequence. The c_j represents a complex number composed of real elements of binary digit j. $m = 6$, $d = 3$.

					$k_5\, k_4\, k_3 \mid k_2\, k_1\, k_0$	Input
					$k_2\, k_4\, k_3 \mid k_5\, k_1\, k_0$	1st *i*-cycle
					$k_2\, k_4\, k_3 \mid n_5\, k_1\, k_0$	Stage 1
					$k_2\, k_4\, k_3 \mid k_1\, k_0\, n_5$	Transpose
					$k_2\, k_1\, k_3 \mid k_4\, k_0\, n_5$	2nd *i*-cycle
				$k_2\, k_1\, k_3 \mid n_4\, k_0\, 0_5$	$k_2\, k_1\, k_3 \mid c_4\, k_0\, 1_5$	Stage 2
				$k_2\, k_1\, k_3 \mid 0_5\, k_0\, n_4$	$k_2\, k_1\, k_3 \mid 1_5\, k_0\, c_4$	Digit switch
				$k_2\, k_1\, 0_5 \mid k_3\, k_0\, n_4$	$k_2\, k_1\, 1_5 \mid k_3\, k_0\, c_4$	3rd *i*-cycle
			$k_2\, k_1\, 0_5 \mid n_3\, k_0\, 0_4$	$k_2\, k_1\, 0_5 \mid c_3\, k_0\, 1_4$	$k_2\, k_1\, 1_5 \mid n_3\, k_0\, c_4$	Stage 3
			$k_2\, k_1\, 0_5 \mid 0_4\, k_0\, n_3$	$k_2\, k_1\, 0_5 \mid 1_4\, k_0\, c_3$		Digit switch
			$0_4\, k_1\, 0_5 \mid k_2\, k_0\, n_3$	$1_4\, k_1\, 0_5 \mid k_2\, k_0\, c_3$	$n_3\, k_1\, 1_5 \mid k_2\, k_0\, c_4$	4th *i*-cycle
		$0_4\, k_1\, 0_5 \mid n_2\, k_0\, 0_3$	$0_4\, k_1\, 0_5 \mid c_2\, k_0\, 1_3$	$1_4\, k_1\, 0_5 \mid n_2\, k_0\, c_3$	$n_3\, k_1\, 1_5 \mid n_2\, k_0\, c_4$	Stage 4
		$0_4\, k_1\, 0_5 \mid 0_3\, k_0\, n_2$	$0_4\, k_1\, 0_5 \mid 1_3\, k_0\, c_2$			Digit switch
		$0_4\, 0_3\, 0_5 \mid k_1\, k_0\, n_2$	$0_4\, 1_3\, 0_5 \mid k_1\, k_0\, c_2$	$1_4\, n_2\, 0_5 \mid k_1\, k_0\, c_3$	$n_3\, n_2\, 1_5 \mid k_1\, k_0\, c_4$	5th *i*-cycle
	$0_4\, 0_3\, 0_5 \mid n_1\, k_0\, 0_2$	$0_4\, 0_3\, 0_5 \mid c_1\, k_0\, 1_2$	$0_4\, 1_3\, 0_5 \mid n_1\, k_0\, c_2$	$1_4\, n_2\, 0_5 \mid n_1\, k_0\, c_3$	$n_3\, n_2\, 1_5 \mid n_1\, k_0\, c_4$	Stage 5
	$0_4\, 0_3\, 0_5 \mid 0_2\, k_0\, n_1$	$0_4\, 0_3\, 0_5 \mid 1_2\, k_0\, c_1$				Digit switch
$0_4\, 0_3\, 0_5 \mid 0_2\, n_0\, 0_1$	$0_4\, 0_3\, 0_5 \mid 0_2\, c_0\, 1_1$	$0_4\, 0_3\, 0_5 \mid 1_2\, n_0\, c_1$	$0_4\, 1_3\, 0_5 \mid n_1\, n_0\, c_2$	$1_4\, n_2\, 0_5 \mid n_1\, n_0\, c_3$	$n_3\, n_2\, 1_5 \mid n_1\, n_0\, c_4$	Stage 6
$0_4\, 0_3\, 0_5 \mid 0_2\, 0_1\, n_0$	$0_4\, 0_3\, 0_5 \mid 0_2\, 1_1\, c_0$					Digit switch
$0_4\, 0_3\, 0_5 \mid 0_2\, 0_1\, n_0$	$0_4\, 0_3\, 0_5 \mid 0_2\, 1_1\, c_0$	$0_4\, 0_3\, 0_5 \mid 1_2\, n_2\, c_1$	$0_4\, 1_3\, 0_5 \mid n_1\, n_0\, c_2$	$1_4\, n_2\, 0_5 \mid n_1\, n_0\, c_3$	$n_3\, n_2\, 1_5 \mid n_1\, n_0\, c_2$	Output

In the second i-cycle the $(m-2)$nd digit is the target. Because another k digit has moved to the processor label, the total number of i-cycles needed for the transform is at least $d+2$. State 2 is then performed on all processors as described above. This is a combination of operations A and B. The complex elements formed are then positioned contiguously in the last half of the distributed array using a local left-most/right-most digit switch. This positions n_{m-1} in the pivot place; hence the maximum number of i-cycles will remain at $d+2$ after the next i-cycle.

The third i-cycle brings k_{m-3} into the pivot. In the third stage, all processors with $n_{m-1} = 1$ perform a third CT1 stage on their subsequence. The other processors perform an operation that is similar to the second stage including the local digit switch.

To make this stage efficient on a SIMD multiprocessor, we must make the combined operations A and B look similar to a CT1 operation. As is evident, operation A is performed on all even elements of the subsequence. This is similar to a CT1 operation with roots (1., 0.). Operation B, which is executed on all odd elements, does not conform to a CT1 operation. One way of accomplishing this is to change the CT1 operation

$$\text{(4.5)} \qquad \text{from } x + \omega^n y \quad \text{to } Wx + \omega^n y,$$

where W is a real array. The roots-of-unity array and W can then be set so that each processor, given the same instruction, will perform either a CT1 operation or an A/B operation. The local digit switch, however, must be a masked operation.

The fourth to the dth stages proceed similarly to the previous stage. On the ith stage, only those processors with $n_{m-1} = n_{m-2} = \cdots = n_{i+2} = 0$ perform an operation equivalent to stage 2. These stages are not seen in Table 4 since $d = 3$.

The next two stages also require i-cycles before they can be executed. The i-cycles reposition k_{m-d-1} and k_{m-d-2} into the pivot. On the $(d+2)$nd stage, only processor 0 performs the stage 2 operation. For the remaining stages, $d+3$ to m, no communication is required. Processor 0 executes an $(m-d-2)$-stage CT1 transform, and the others complete their stages. Figure 1 shows a graph of the complete CT1R algorithm for

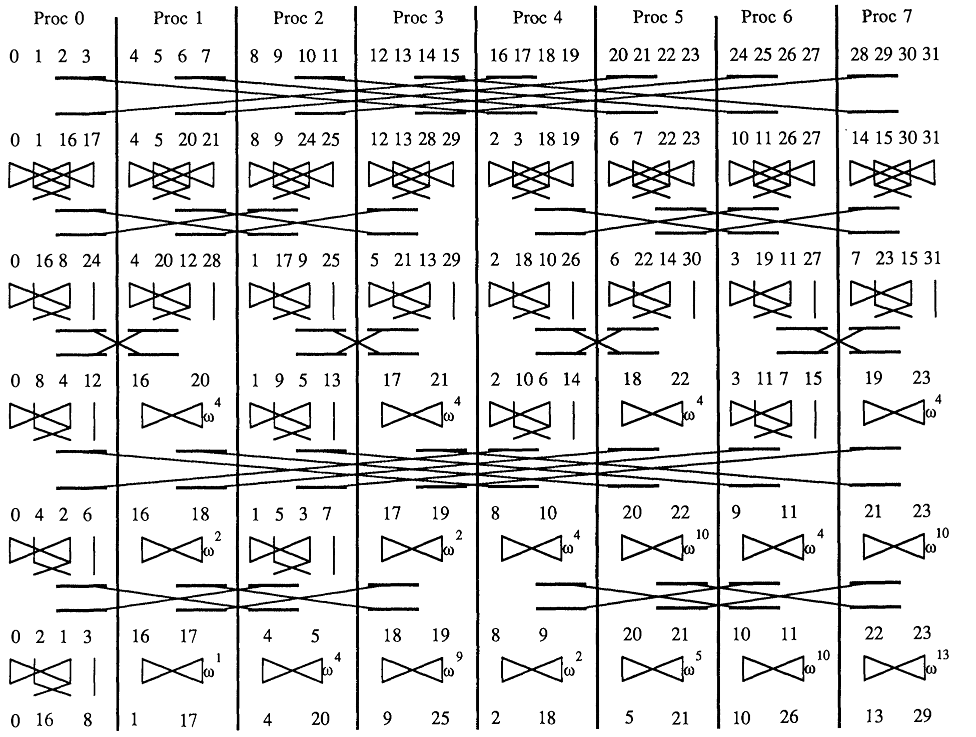


FIG. 1. *Graph of the* CT1R *algorithm with* $m = 5$, $d = 3$. *The input is a real, naturally ordered sequence and the output is a complex, bit-reversed, second-conditionally ordered, processor transposed sequence. Operation* A *is executed on the first stage on the first and third, and second and fourth elements. The transpose is shown just below that. The second stage involves operation* A *on even elements and* B *on odd. The digit switch is shown just below. On the third stage, the odd processors execute the complex* CT1 *operation, shown as a single butterfly with a root of unity.*

$m = 5$, $d = 3$. This completes the description of the transform; there is a total of $d+2$ i-cycles.

4.3. Ordering, roots, and inverse transform. In addition to the output sequence being in the second-conditional, bit-reversed order, the processor index is not in the proper bit-reversed order. The two left-most digits should be on the right. It requires $d+2$ i-cycles and local permutations to correct this.

For the CT1R algorithm we have chosen to precompute the roots of unity. For stages 3 to $d+2$ with $d \geq 2$, each processor requires one root per stage. For the pth processor, we first form p_0 by

$$p_0 = 4p/P + 4(p)_{\bmod P/4}, \tag{4.6}$$

which is the adjusted processor label. Let j be the first nonzero binary digit from the left in p_0. Then the adjusted processor label is modified so that the 1 in the jth place is moved one place to the left,

$$p_1 = 2^{j+1} + (p_0)_{\bmod 2^j}. \tag{4.7}$$

The argument for root of unity is then

$$\tfrac{1}{4} 2\pi/P \text{ bit-reverse } (p_1 \text{ with } d+1 \text{ places}). \tag{4.8}$$

The roots from stages $d+3$ to m build on the roots from stage $d+2$ in the standard way.

The inverse transform can be accomplished by reversing the stages, i-cycles, and local permutations. The input data should be in the same order as the output of the forward transform. As in the serial case, this corresponds to the GS2CS algorithm.

4.4. Performance analysis. In order to estimate the parallel performance of the algorithm presented above, we first must establish the operation count for each processor during each stage. We shall analyze the load imbalance that occurs at each stage and note the amount of local memory transfers. We shall denote the number of transforms by q. It is then a simple matter to calculate the amount of communication overhead. Together with the communication time, we shall predict such measures as overhead and parallel efficiency.

In the first and second stages the computations are completely balanced; the total number of floating-point operations is $\frac{3}{2}qN/P$ for each processor. There are $(m-d-1)qN/2P$ elements exchanged in the local transpose and $qN/2P$ elements exchanged in the local digit switch. For stages 3 to $d+2$, processors either perform the real summation with an operation count of $\frac{1}{2}qN/P$ or the CT1 operation with a count of $5qN/2P$. In stages $d+3$ to m the total operation count is $5q(m-d-2)N/2P-2qN/P$ for processor 0 and $5q(m-d-2)N/2P$ for the others.

Totaling only the maximum operation count for each stage and ignoring the local memory exchanges, the operation count for the transform is

$$T_P^{\text{calc}} = (5m-2)Nq/2P, \tag{4.9}$$

which is similar to the operation count for a complex FFT of length $N/2$.

$d+2$ i-cycles are required for the transform; hence there are $d+2$ nearest-neighbor communication exchanges of contiguous data of length $Nq/2P$. The equivalent number of floating-point operations required by the i-cycles is then

$$T_P^{\text{comm}} = 2(d+2)\left(\alpha+\beta\frac{Nq}{2P}\right), \tag{4.10}$$

where α is the number of floating-point operations per communication startup, and β is the number of floating-point operations per word transmitted.

Overhead is defined as

$$f=(PT_P-T_1)/T_1, \tag{4.11}$$

where T_1 is the number of operations required for the transform run on one processor. The operation count for the serial algorithm is $(5m-2)Nq/2$. The overhead for the real transform is then

$$f=\frac{5}{2}\frac{d+2}{m-2}\left(\beta+\alpha\frac{2P}{Nq}\right). \tag{4.12}$$

The first term in the parentheses comes from the message-passing communication. The last is message startup and is inversely proportional to the granularity $Nq/2P$. It will most likely be small when granularity is large. Multiplying the expression is approximately $\log P/\log N$. The overhead is proportional to the dimension and inversely proportional to m.

The efficiency is $[1+f]^{-1}$. Neglecting startup and taking the first term of the binomial series, it can be expressed as

$$e\sim 1-\frac{2}{5}\frac{d+2}{m-2}\beta. \tag{4.13}$$

An efficiency close to one is expected for problems with large granularity and multiprocessors with small β (ratio of message passing to computational rates).

4.5. A serial GS1R algorithm. An attraction of the GS1 forward transform is that it has a less complicated power distribution for the roots-of-unity multiplication than does the CT1 transform. The former has sequentially ordered roots while the latter has bit-reversed ordered roots. For completeness and simplicity, we present a discussion of a parallel GS1R algorithm for a real, naturally ordered sequence in this and the next two sections. We have chosen this particular way of implementing the GS1R for ease in parallelization.

As in the CT1R algorithm, the first step involves only the k_{m-1} summation, which is operation A. In the output, k_{m-1} is replaced by n_{m-1}.

The second step is split into two parts. For input data with $n_{m-1}=0$, the ω_N term is unity and the k_{m-2} summation (operation A) is done. The index of the output is $(n_{m-1}=0\ n_{m-2}\ k_{m-3}\dots k_0)$ and the data are real. For $n_{m-1}=1$, we combine the roots calculation with the k_{m-2} summation. Since the input sequence is real, the output with $n_{m-2}=1$ are the complex conjugates of the corresponding output data with $n_{m-2}=0$; hence, the former output data are not calculated. The $\omega_{N/2}$ term is unity for the output calculated. This is operation B with a roots-of-unity multiplication.

The third stage is split into three parts. For the real subsequences with $n_{m-1}=n_{m-2}=0$, the k_{m-3} summation is performed and the output is real. For $n_{m-1}=0$ and $n_{m-2}=1$, we have a situation similar to the second part of stage 2. The last part of stage 3 is for complex input sequences with $n_{m-1}=1$; the GS1 stage is executed.

Stages 4 to m proceed similarly to the proceeding stage. If all n digits to the left of the current place are zero, then the summation of real input to real output is performed. If all n digits to the left are zero except the adjacent one, then the real-to-complex operation is executed and is similar to stage 2. In all other cases, the GS1 operation stage is executed.

The output vector has the second-conditional, bit-reversed ordering; the inverse is the CT2CS algorithm.

4.6. The parallel GS1R. The previous section provided a description of the GS1R algorithm for a real, naturally ordered sequence. Now we develop the parallel version.

Table 4 contains the index representations at each stage and i-cycle for the parallel GS1R (as well as CT1R) for $m=6$ and $d=3$. All the communications and local digit permutations are the same as the CT1R algorithm; the only differences are the roots of unity and the operations at each state. Figure 2 shows a graph of the GS1R with $m=5$, $d=3$.

Stage 1 is operation A. Stage 2 is in two parts. If $n_{m-1}=0$, then just the operation A is executed. If $n_{m-1}=1$, then the complex number is created (operation B) and then multiplied by the suitable root of unity. The local digit switch is then done.

The third i-cycle brings the $(m-3)$rd place to local memory and moves n_{m-1} or q_{m-2} to the processor label. In all subsequent i-cycles, only n or q indices will move into the processor label. Thus, the number of i-cycles for the transform stands at $d+2$. The third stage is split into three parts. If $n_{m-1}=n_{m-2}=0$, then the simple real summation is performed. If $n_{m-1}=0$ and $n_{m-2}=1$, then the complex number/roots-of-unity operation is done followed by a reindexing with q replacing n. If $q_{m-2}=1$, then a standard GS operation is done. A local index switch is done on those processors with $n_{m-1}=0$.

The subsequent stages 4 through $d+2$ proceed similarly. Stages $d+3$ to m and index switches proceed as shown in the table. The resulting index order is shown in

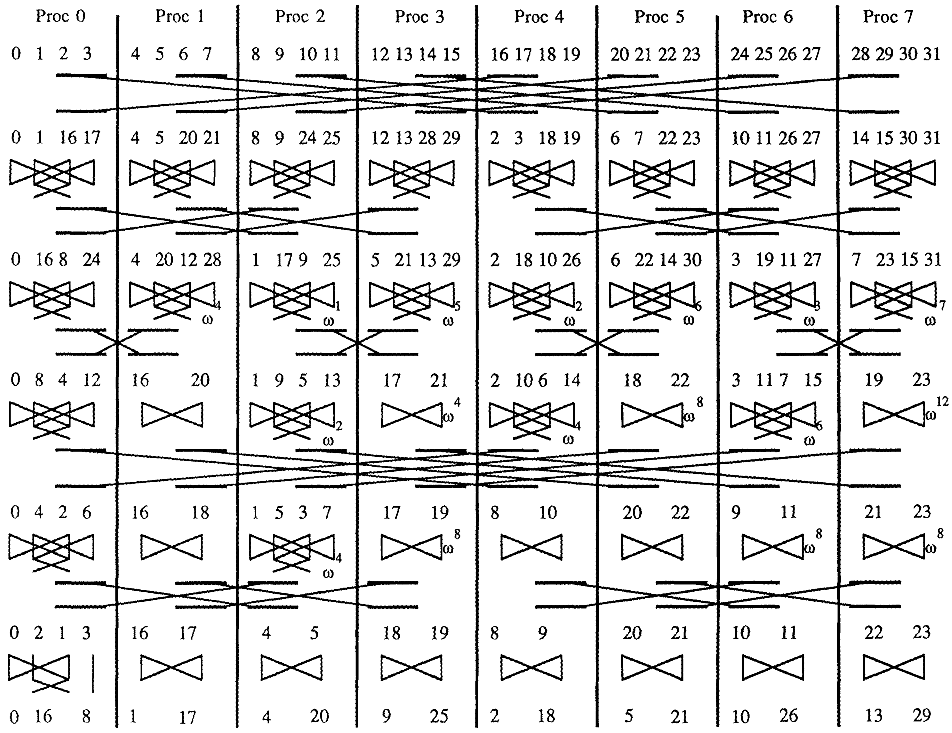


FIG. 2. *Graph of the* GS1R *algorithm with $m=5$, $d=3$. The input is a real, naturally ordered sequence and the output is a complex, bit-reversed, second-conditionally ordered, processor transposed sequence. The first stage is the same as in Fig.* 1. *On the second stage, even elements execute* A *and odd execute* B *plus a roots-of-unity multiplication. On subsequent stages the single butterfly with root represents a complex* GS1 *operation.*

the last entry of the table. This completes the development of the parallel GS1R algorithm.

4.7. Ordering, roots, and the inverse transform. The ordering of the output data is the same as in the parallel CT1R algorithm.

We have chosen to compute the roots of unity in the standard way by precomputing the first root, or the increment for each stage,

$$\delta\omega_j = \exp(i\pi/2^{m-j}), \quad \text{where } j \text{ is the stage index.} \tag{4.14}$$

During each stage, the actual roots are calculated. With this method, the starting root for each processor is not necessarily $(1,0)$ and so a starting root of unity; $\omega_j^0(k)$ must also be precomputed for each processor and each stage.

One way of computing the starting roots of unity for $d>1$ is

$$\begin{aligned}
\omega_1^0(k) &= \exp[i(k_{m-3}, k_{m-4}, \dots, k_{m-d-2})\pi/2P],\\
\omega_2^0(k) &= \exp[i(k_{m-3}, k_{m-4}, \dots, k_{m-d-2})\pi/P],\\
\omega_3^0(k) &= \exp[i(k_{m-4}, \dots, k_{m-d-2}, 0)\pi/P],\\
\omega_4^0(k) &= \exp[i(k_{m-3}, \dots, k_{m-d-2}, 0, 0)\pi/P],\\
&\ \ \vdots\\
\omega_{d+1(k)}^0(k) &= \exp[i(k_{m-d-2}, 0, \dots, 0)\pi/P],\\
\omega_j^0(k) &= (1,0) \text{ for } j=d+2 \text{ to } m.
\end{aligned} \tag{4.15}$$

If $d=1$, then

$$\omega_1^0(k)=\exp[1(m-2)\pi/2P], \qquad \omega_j^0(k)=(1,0) \quad \text{for } j=2 \text{ to } m. \tag{4.16}$$

The inverse transform can be accomplished by reversing the stages and operations. The input data should be in the same order as the output of the forward transform. This corresponds to the CT2CS algorithm.

4.8. Performance analysis. The first and second stages have completely balanced computations; the total number of floating-point operations is Nq/P and $N/2P(5q+3)$ for each processor, respectively. For stages 3 to $d+2$, processors also have a balanced computational load of $N/2P(5q+3)$. The total operation count for stages $d+3$ to m is $N/2P[(m-d-2)5q+6]$ for processor 0 and $N/2P[(m-d-2)5q+3]$ for all others. Since processor 0 has slightly more work and the local permutation, an imbalance may occur. Totaling the maximums for each stage, the operation count for the transform is

$$T_P^{\text{calc}}=N/2P[(m-3/5)5q+3d+9]. \tag{4.17}$$

The second term in parentheses comes from redundant roots-of-unity calculations during stages 2 to $d+2$.

There are $d+2$ nearest-neighbor communication exchanges of contiguous data of length $n/2P$. The communication time is

$$T_P^{\text{comm}}=2(d+2)\left(\alpha+\beta\frac{Nq}{2P}\right). \tag{4.18}$$

The operation count for the serial algorithm is $N/2[(m-3/5)5q+P]$. The overhead for the real transform is then

$$f=\frac{[3d/q+2(d+2)\alpha][2P/Nq+2(d+2)\beta]}{(m-3/5)+9/q}. \tag{4.19}$$

The first term in the numerator comes from the redundant roots-of-unity calculations and is small for large q. The second term is from message startup and is inversely proportional to the granularity $Nq/2P$. The third term is from the message-passing communication and is proportional to the dimension and message-passing rate. For large granularity, defined as $\alpha/\beta \ll Nq/2P$, and large, q, $q \gg d$, the overhead tends to 2β d/m.

The efficiency is $[1+f]^{-1}$. Taking the first term of the binomial expansion, the efficiency is

$$e \sim 1-2\beta\ \text{d/m} \tag{4.20}$$

for the assumptions made above.

5. The parallel, compact FFT for a conditionally ordered, conjugate-symmetric sequence.

In this section we present the parallel compact CT1CS and GS1CS algorithms. The conjugate-symmetric input sequence is conditionally ordered and the real output sequence is in bit-reversed order.

5.1. The parallel GS1CS algorithm. To describe the algorithm, we first describe the inverse transform CT2R, which has as its input, a real, bit-reversed sequence. Much of this algorithm is the same as CT1R; the reader is referred to §§ 4.1 and 4.2 for details of the operations at each stage.

Let the binary index of the real sequence be

$$(n_0\, n_1 \ldots n_{m-1}). \tag{5.1}$$

The index of the distributed, bit-reversed sequence is then

$$(n_{m-1}\, n_{m-2} \dots n_{m-d} \,|\, n_{m-d-1}\, n_{m-d-2} \dots n_0). \tag{5.2}$$

We also assume that the input sequence has the processor transposition [13]; the index of the input sequence is then

$$(n_{m-d}\, n_{m-1}\, n_{m-2} \dots n_{m-d+1} \,|\, n_{m-d-1}\, n_{m-d-2} \dots n_0). \tag{5.3}$$

Table 5 has an example of the index configurations in the transform for $m=6$, $d=3$. The first stage of the algorithm is the summation over n_0, which is operation A described in § 2. The second stage is a summation over n_1. If $k_0=0$, then operation A is executed; if $k_0=1$, then operation B is executed and a complex number is created from the two real elements. The complex number is denoted by a c_1 in the table. The indices k_1 and k_0 are switched.

TABLE 5

Binary index table of the inverse compact transform with a real, bit-reverse-ordered input sequence and conjugate symmetric, naturally ordered output sequence. The c_j represents a complex number composed of real elements of binary digit j. $m=6$, $d=3$.

					$n_3 n_5 n_4 \vert n_2 n_1 n_0$	Initial index
					$n_3 n_5 n_4 \vert n_2 n_1 k_0$	Stage 1
				$n_3 n_5 n_4 \vert n_2 k_1 0_0$	$n_3 n_5 n_4 \vert n_2 c_1 1_0$	Stage 2
				$n_3 n_5 n_4 \vert n_2 0_0 k_1$	$n_3 n_5 n_4 \vert n_2 1_0 c_1$	Digit switch
			$n_3 n_5 n_4 \vert k_2 0_0 0_1$	$n_3 n_5 n_4 \vert c_2 0_0 1_1$	$n_3 n_5 n_4 \vert k_2 1_0 c_1$	Stage 3
			$n_3 n_5 n_4 \vert 0_1 0_0 k_2$	$n_3 n_5 n_4 \vert 1_1 0_0 c_2$		Digit switch
			$0_1 n_5 n_4 \vert n_3 0_0 k_2$	$1_1 n_5 n_4 \vert n_3 0_0 c_2$	$k_2 n_5 n_4 \vert n_3 1_0 c_1$	1st i-cycle
		$0_1 n_5 n_4 \vert k_3 0_0 0_2$	$0_1 n_5 n_4 \vert c_3 0_0 1_2$	$1_1 n_5 n_4 \vert k_3 0_0 c_2$	$k_2 n_5 n_4 \vert k_3 1_0 c_1$	Stage 4
		$0_1 n_5 n_4 \vert 0_2 0_0 k_3$	$0_1 n_5 n_4 \vert 1_2 0_0 c_3$			Digit switch
		$0_1 n_5 0_2 \vert n_4 0_0 k_3$	$0_1 n_5 1_2 \vert n_4 0_0 c_3$	$1_1 n_5 k_3 \vert n_4 0_0 c_2$	$k_2 n_5 k_3 \vert n_4 1_0 c_1$	2nd i-cycle
	$0_1 n_5 0_2 \vert k_4 0_0 0_3$	$0_1 n_5 0_2 \vert c_4 0_0 1_3$	$0_1 n_5 1_2 \vert k_4 0_0 c_3$	$1_1 n_5 k_3 \vert k_4 0_0 c_2$	$k_2 n_5 k_3 \vert k_4 1_0 c_1$	Stage 5
	$0_1 n_5 0_2 \vert 0_3 0_0 k_4$	$0_1 n_5 0_2 \vert 1_3 0_0 c_4$				Digit switch
	$0_1 0_3 0_2 \vert n_5 0_0 k_4$	$0_1 1_3 0_2 \vert n_5 0_0 c_4$	$0_1 k_4 1_2 \vert n_5 0_0 c_3$	$1_1 k_4 k_3 \vert n_5 0_0 c_2$	$k_2 k_4 k_3 \vert n_5 1_0 c_1$	3rd i-cycle
$0_1 0_3 0_2 \vert k_5 0_0 0_4$	$0_1 0_3 0_2 \vert c_5 0_0 1_4$	$0_1 1_3 0_2 \vert k_5 0_0 c_4$	$0_1 k_4 1_2 \vert k_5 0_0 c_3$	$1_1 k_4 k_3 \vert k_5 0_0 c_2$	$k_2 k_4 k_3 \vert k_5 1_0 c_1$	Stage 6
$0_1 0_3 0_2 \vert 0_4 0_0 k_5$	$0_1 0_3 0_2 \vert 1_4 0_0 c_5$					Digit switch
$0_4 0_3 0_2 \vert 0_1 0_0 k_5$	$1_4 0_3 0_2 \vert 0_1 0_0 c_5$	$k_5 1_3 0_2 \vert 0_1 0_0 c_4$	$k_5 k_4 1_2 \vert 0_1 0_0 c_3$	$k_5 k_4 k_3 \vert 1_1 0_0 c_2$	$k_5 k_4 k_3 \vert k_2 1_0 c_1$	4th i-cycle

The third to $(m-d)$th stages proceed as follows. On the jth stage, operation A is executed if all k bits are zero. Operation B is executed if all k bits are zero except k_{j-2}. Complex CT2 operations are executed on the complement. In a local memory exchange, indices k_{j-2} and k_{j-1} are switched.

Before the $(m-d-1)$st stage, an i-cycle places the right-most target bit into the pivot position. On the $(m-d-1)$st stage, the lower half of the processors execute a combined A/B operation, and the upper half execute CT2 operations.

For the next $d-1$ stages, $m-d+2$ to m, we first perform i-cycles with the target starting from the right-most processor bit and moving progressively to the left with each stage. Those processors with zero k bits execute operation A/B and the local digit switch, while the complement execute the CT2 operations. Finally, after an i-cycle with the right-most bit as the target is done, the transform is complete and the complex output sequence is in conditional order.

Figure 3 shows the graph of the GS1CS transform for $m=5$ and $d=3$. The complex input, which is part of the conjugate-symmetric sequence, is in conditional order. The first i-cycle has the right-most processor bit as the target. Processor 0 must execute an operation $(\mathrm{A/B})^{-1}$ and the others a GS1 operation. The resulting index of

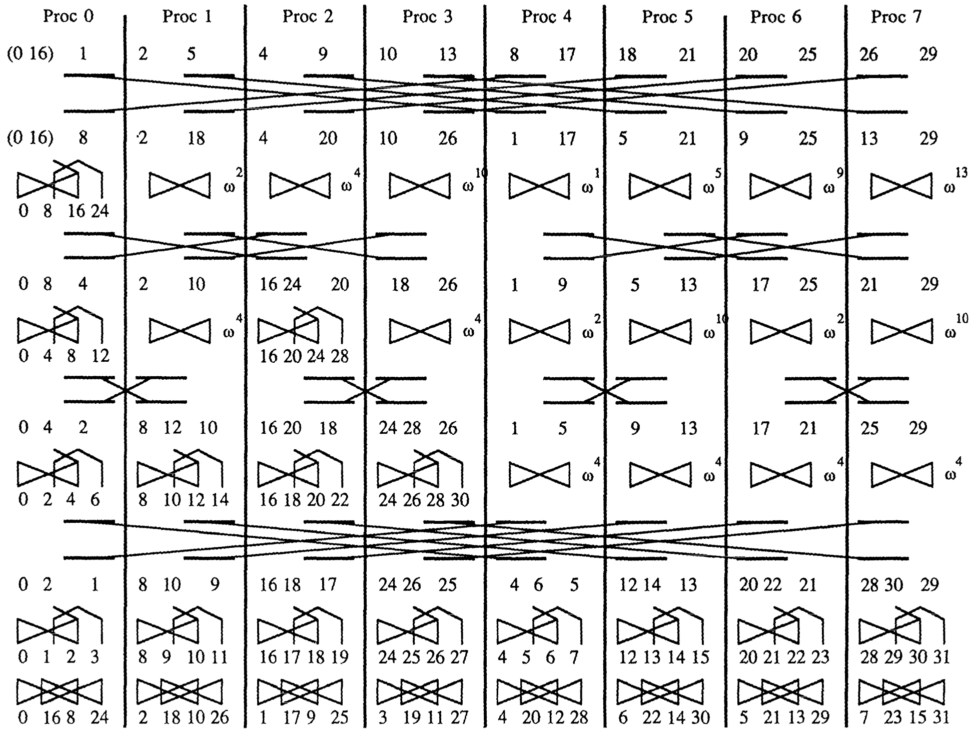


FIG. 3. *Graph of the* GS1CS *algorithm with $m=5$, $d=3$. The input is a complex, conditionally ordered sequence, and the input is a real, bit-reversed, processor-transposed sequence. The zero processor executes a digit switch and* A^{-1} *on the even elements and* B^{-1} *on the odd. The rest of the processors execute complex* GS1 *operations. On the final stage, operation* A^{-1} *is executed on first and third, second, and fourth elements.*

real elements is shown. Note that first a digit switch is done to make the $(A/B)^{-1}$ operation have a similar form to that of GS1. Operations $(A/B)^{-1}$ and GS1 can be combined as in (4.5) for SIMD multiprocessors. The argument for the starting roots of unity is

$$2(p_{\mathrm{mod}\,P/2})+\frac{p}{P/2}, \tag{5.4}$$

where p is the base-10 processor index.

The second i-cycle has the middle processor bit as the target. We define a reduced processor index on stage j as

$$p_j=p_{j-1}-2^{d-j}, \tag{5.5}$$

where $p_1=p$. All processors with positive reduced processor index execute GS1 operations; the complement execute operation $(A/B)^{-1}$. Hence, on the second stage, processors 0 and 2 execute $(A/B)^{-1}$. The argument for the roots on the jth stage is

$$2\pi 2^{j-1}(p_{\mathrm{mod}\,P/2})/2N. \tag{5.6}$$

The target in the third i-cycle is the right-most processor bit. The lower half of the processors execute $(A/B)^{-1}$ and the upper half execute GS1. The final i-cycle is the same as the first. All processors execute $(A/B)^{-1}$ on the fourth stage and A^{-1} on the fifth. The last line has the index of the bit-reversed real sequence.

This completes the description of the algorithm. Only $d+1$ i-cycles are necessary to transform a conjugate-symmetric sequence in conditional order into a real, m bit-reversed, processor-transposed sequence. The starting roots of unity for each processor for each stage are given in (5.4) and (5.6), and the increment is given in (4.14). The transform CT2R is obtained by formally inverting the GS1CS operations and i-cycles. The parallel performance is similar to the GS1R algorithm. See § 4.6 for more details.

5.2. The parallel CT1CS algorithm. We have presented three parallel algorithms (and their inverses) for the transform of real data. In the first two, a real naturally ordered sequence was input, and a complex sequence of bit-reversed, second-conditional ordering was the output. The third, based on the compact GS1 algorithm, has a complex input sequence in conditional ordering and a real bit-reversed output sequence. The final transform has the same ordering of sequences as the third, but is based on the CT1 algorithm.

The evolution of the binary indices is the same as in the previous transform and is best explained using the inverse transform. Table 5, which was used to illustrate GS1CS, is also valid for CT1CS.

Figure 4 shows a graph of the forward transform for $m=5$, $d=3$. The scheme follows the GS1R scheme in reverse except that the sequences are in a different ordering. The CT1CS has a conjugate-symmetric, conditionally ordered input sequence and a real, bit-reversed output sequence, whereas the GS1R scheme has a real, naturally ordered input sequence and a conjugate-symmetric, second-conditionally ordered, bit-reversed output sequence. Roots of unity can be obtained from (4.6) to (4.8). We chose to precompute the roots of unity; the operation count is the same as in the CT1R algorithm (4.9). The inverse of this transform is GS2R.

6. Results. To complete the presentation of parallel compact FFTs for real sequences, we give timings of four of the eight algorithms as implemented on the nCUBE/2, an MIMD multiprocessor with version 3.0beta of the operating system. All code is written in FORTRAN-77 with explicit calls to the communication routines and run using single precision (32 bit words).

A 64-processor machine was employed for the tests. We chose to report one sequence length, 4096, with the number of sequences being 1, 8, and 16. With these parameters all the tests could be run on one processor (4 Mbytes RAM per processor), so that parallel efficiencies could be computed. The scheme that was run on one processor is the same as was run on multiprocessors. The transpose and index switch, not necessary for the single-processor run, was nonetheless executed. Due to limitations on the communications buffer, a few low-dimension tests could not be run. All results reported are an average of times over all processors. The timings are taken from a forward/inverse pair after many pairs have been executed.

Tables 6(a), 6(b), and 6(c) show the times in milliseconds for the CT1R (forward) transform and the GS2CS (inverse) transform. The first column is the processor dimension, which is the log (base 2) of the number of processors used to calculate the transform. The other columns from left to right are the total time for one transform, the calculation time, the communication time including idle time, the time taken in the transpose and index switches, and the parallel efficiency. These columns are repeated for the inverse transform.

Immediately obvious is that the time for index switch, which consists of local memory references only, takes from 37% to 14% of the total time, depending superlinearly on the processor number. In order for these algorithms to be competitive with

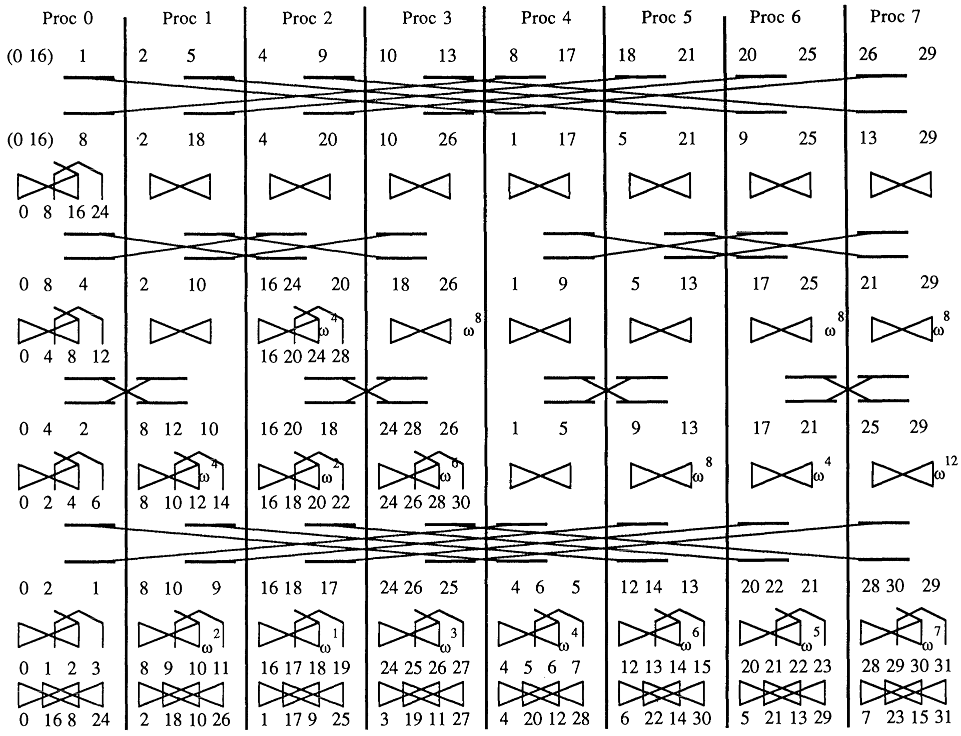


FIG. 4. *Graph of the* CT1CS *algorithm with $m = 5$, $d = 3$. The input is a complex, conditionally ordered sequence, and the input is a real, bit-reversed, processor-transposed sequence. The first and last stages are the same as in Fig.* 3. *On the second stage processor* 2 *executes a digit switch, roots-of-unity multiplication and* B^{-1}.

CLW, local memory referencing must be done more efficiently; transpose and index switch routine should be coded in assembler.

The calculation time diverges from linear scaling as the dimension increases. Loop overhead is thought to be responsible for this behavior. Since the roots of unity are precomputed and stored in this implementation, the nonlinear (in P) term in (4.17) is not present in the operation count. The less-than-linear scaling of the communication time reflects the fact that message startup time scales as the dimension and the message-passing time scales as the dimension/P (see (4.18)). An efficiency of 58% is obtained on 64 processors and a subsequence length of 64.

Comparing the inverse transform GS2CS to the forward transform in Table 6(a), the higher total time is due to the normalization performed on the inverse, which increases the calculation time. The communication time is higher also. The reason for this is that the forward transform, executed first, sets a certain processor synchronization, which causes an increased idle time on the inverse. The transpose and index switch times of the inverse are essentially the same as those of the forward. Efficiencies of the inverse are slightly lower than those of the forward.

Table 6(b) contains information similar to Table 6(a), except that eight sequences are transformed instead of one. An example of loop overhead can be seen in the comparison of the dimension 0 calculation times of Tables 6(a) and 6(b). There is a factor of 4.6 instead of 8 between them. The calculation and communication times scale more linearly with processor number for eight transforms. Transpose and index switch times still scale superlinearity. An efficiency of 77% is achieved on 64 processors.

TABLE 6(a)
Averaged milliseconds per Cooley-Tukey FFT *of a real, naturally ordered sequence (forward transform) and the Gentleman-Sande* FFT *of a conjugate-symmetric, bit-reverse ordered sequence (inverse transform). Sequence length*: $N = 4096$; *number of sequences*: $q = 1$.

	CT1R forward transform					GS2CS inverse transform				
Dim	Total	Calc	Comm	Sw	Eff	Total	Calc	Comm	Sw	Eff
0	279.94	173.44	0.00	104.83	1.00	290.69	182.14	0.00	107.26	1.00
1	144.83	86.72	8.32	48.58	0.97	151.49	92.35	8.70	49.73	0.96
2	72.99	43.71	5.79	22.46	0.96	77.57	46.37	7.33	22.85	0.94
3	37.46	22.06	4.08	10.38	0.93	40.80	23.33	5.92	10.50	0.89
4	20.01	11.30	3.06	4.79	0.87	22.53	11.77	4.89	4.79	0.81
5	11.58	5.69	2.48	2.20	0.76	13.59	5.97	4.27	2.18	0.67
6	7.60	3.04	2.47	1.04	0.58	9.41	3.21	4.13	1.04	0.48

TABLE 6(b)
Averaged milliseconds per Cooley-Tukey FFT *of a real, naturally ordered sequence (forward transform) and the Gentleman-Sande* FFT *of a conjugate-symmetric, bit-reverse ordered sequence (inverse transform). Sequence length*: $N = 4096$; *number of sequences*: $q = 8$.

	CT1R forward transform					GS2CS inverse transform				
Dim	Total	Calc	Comm	Sw	Eff	Total	Calc	Comm	Sw	Eff
0	1108.35	798.98	0.00	308.22	1.00	1166.21	858.24	0.00	306.18	1.00
2	297.89	190.62	41.15	65.25	0.93	323.68	215.52	42.34	64.86	0.90
3	150.50	95.17	24.74	29.58	0.92	166.64	108.08	28.16	29.39	0.87
4	77.74	47.86	15.54	13.38	0.89	85.78	54.14	17.37	13.29	0.85
5	41.05	24.29	9.79	5.95	0.84	44.92	27.33	10.64	5.92	0.81
6	22.60	12.35	6.47	2.69	0.77	24.58	13.89	6.97	2.67	0.74

TABLE 6(c)
Averaged milliseconds per Cooley-Tukey FFT *of a real, naturally ordered sequence (forward transform) and the Gentleman-Sande* FFT *of a conjugate-symmetric, bit-reverse ordered sequence (inverse transform). Sequence length*: $N = 4096$; *number of sequences*: $q = 16$.

	CT1R forward transform					GS2CS inverse transform				
Dim	Total	Calc	Comm	Sw	Eff	Total	Calc	Comm	Sw	Eff
0	2055.30	1513.73	0.00	540.42	1.00	2164.86	1626.88	0.00	536.83	1.00
3	280.27	179.12	48.42	51.74	0.92	312.85	203.97	56.37	51.41	0.86
4	143.82	89.92	29.70	23.29	0.89	160.28	102.26	33.98	23.00	0.84
5	74.78	45.41	18.03	10.32	0.86	82.92	51.40	20.22	10.24	0.82
6	39.80	23.04	11.15	4.55	0.81	43.77	25.96	12.27	4.53	0.77

The calculation and communication times are slightly higher for the inverse due to the same reasons as in Table 6(a).

Timings for the transform of 16 sequences are shown in Table 6(c). Comparing calculation times of Tables 6(b) and 6(c), it is seen that loop overhead is small and that scaling is nearly linear with the processors' number. Memory references still take a significant amount of the total time. Efficiency is 81% for 64 processors. The timings for the inverse have a trend similar to Table 6(b).

Tables 7(a), 7(b), and 7(c) show the times in milliseconds for the GS1R (forward) transform and the CT2CS (inverse) transform for 1, 8, and 16 sequences of length 4096. The columns are arranged similarly to those in Table 6(a), 6(b), and 6(c). The roots of unity are not precomputed, although the root increments are precomputed and stored.

As can be seen in the comparison of Table 6(a) to Table 7(a), the calculation times for the GS1R are higher than the CT1R. This is due to the roots-of-unity

TABLE 7(a)

Averaged milliseconds per Gentleman-Sande FFT *of a real, naturally ordered sequence (forward transform) and the Cooley-Tukey* FFT *of a conjugate-symmetric, bit-reverse ordered sequence (inverse transform). Sequence length*: $N = 4096$; *number of sequences*: $q = 1$.

	GS1R forward transform					CT2CS inverse transform				
Dim	Total	Calc	Comm	Sw	Eff	Total	Calc	Comm	Sw	Eff
0	342.91	236.54	0.00	105.22	1.00	320.51	211.71	0.00	106.88	1.00
1	177.02	119.04	8.19	48.83	0.97	176.51	107.39	18.30	49.60	0.91
2	89.60	60.06	5.70	22.53	0.96	95.17	54.53	16.67	22.85	0.84
3	46.29	30.72	4.06	10.38	0.93	52.66	28.02	13.18	10.54	0.76
4	24.75	15.82	3.10	4.76	0.87	30.13	14.42	9.91	4.80	0.66
5	14.06	8.15	2.52	2.15	0.76	18.16	7.34	7.32	2.13	0.55
6	8.95	4.34	2.44	1.04	0.60	12.01	3.92	5.96	1.04	0.42

TABLE 7(b)

Averaged milliseconds per Gentleman-Sande FFT *of a real, naturally ordered sequence (forward transform) and the Cooley-Tukey* FFT *of a conjugate-symmetric, bit-reverse ordered sequence (inverse transform). Sequence length*; $N = 4096$; *number of sequences*: $q = 8$.

	GS1R forward transform					CT2CS inverse transform				
Dim	Total	Calc	Comm	Sw	Eff	Total	Calc	Comm	Sw	Eff
0	1322.62	1011.20	0.00	309.63	1.00	1263.36	954.75	0.00	307.33	1.00
2	358.30	250.24	41.09	65.70	0.92	364.99	238.72	60.22	65.12	0.87
3	181.30	124.43	26.02	29.87	0.91	192.30	118.75	42.82	29.66	0.82
4	92.38	61.66	15.97	13.47	0.89	102.47	59.40	28.78	13.35	0.77
5	48.01	30.83	9.98	6.06	0.86	55.87	29.73	19.16	5.99	0.71
6	26.10	15.79	6.45	2.69	0.79	32.36	15.08	13.55	2.67	0.61

TABLE 7(c)

Averaged milliseconds per Gentleman-Sande FFT *of a real, naturally ordered sequence (forward transform) and the Cooley-Tukey* FFT *of a conjugate-symmetric, bit-reverse ordered sequence (inverse transform). Sequence length* : $N = 4096$; *number of sequences*: $q = 16$.

	GS1R forward transform					CT2CS inverse transform				
Dim	Total	Calc	Comm	Sw	Eff	Total	Calc	Comm	Sw	Eff
0	2449.28	1904.13	0.00	543.23	1.00	2345.98	1805.44	0.00	538.37	1.00
3	336.83	232.58	51.09	52.16	0.91	353.04	223.79	76.77	51.57	0.83
4	170.72	115.10	31.00	23.50	0.90	186.17	111.16	50.78	23.20	0.79
5	87.30	57.22	18.63	10.41	0.88	98.99	55.46	32.25	10.33	0.74
6	45.66	28.45	11.44	4.58	0.84	53.91	27.71	20.65	4.54	0.68

calculation. We have chosen to normalize on the forward transform, which also adds to the calculation time. The communication time, transpose and index switch times, and efficiencies are effectively the same as for the CT1R transform.

When compared to the forward transform GS1R, the inverse transform CT2CS has a lower calculation time, which results from no normalization being performed. A higher idle time is reflected in a higher communication time and is caused by a particular processor synchronization, which minimizes the idle time of the forward transform. These two factors combine to create lower parallel efficiencies for the inverse. Similar comparisons are found for all three sequences.

Tables 8(a), 8(b), and 8(c) show timings for transforms of real sequences using the CLW algorithm. We shall briefly review the implementation; see [8] for more details. Walker, Worley, and Drake [16] have implemented the CLW algorithm on the Intel iPSC/860. The real input sequence is treated as a complex sequence following (3.1). The ordering of the complex array is assumed to be bit-reversed. A CT2 transform is then performed on the complex sequence. The roots of unity are not precomputed. The output from the transform is in sequential order to simplify the postprocessing step, (3.2).

Before executing (3.2), the sequence is written into a work array which is then redistributed in index-reversed order. As discussed in § 3.2, the reordering takes at least $d-1$ communications. Because we chose here to completely reorder the sequence, the message length is $(Nq/p-1)$. After the reordering of the work array, (3.2) is executed and the conjugate-symmetric output sequence is in sequential order. The inverse proceeds with a preprocessing step and a GS1 transform of a complex sequence. The suffixes f and p on the column headings of Tables 8(a), 8(b), and 8(c) relate the complex FFT and the pre- and postprocessing, respectively.

Because of the roots-of-unity computation it is best to compare the results of the GS1R/CT2CS transform (Tables 7(a), 7(b), and 7(c)) with the results of the CLW transform (Tables 8(a), 8(b), and 8(c)). Comparing dimension 0 timings in Tables 7(a) and 8(a), we see that the total time for CLW-CT2 is about half that of GS1R. The GS1R calculation time is about 30% higher. Similar trends, though progressively less severe, are seen in the dimension 1 to 5 results. The $d+2$ communications of the GS1R can be compared to the $d+1$ communications of the complex FFT of CLW (commf). Results of dimension 6 show that the total time of the GS1R is less than CLW-CT2. The calculation time of the CLW-CT2 is still less than that of GS1R, but the postprocessing communication has become dominant over the index switch time.

Comparing Table 7(b) to Table 8(b) for the transform of eight sequences, the dimension 0 case shows that the total time for the CLW-CT2 is still lower than the GS1R, but the calculation times are about the same. This latter result is reinforced by estimates of operation count. For dimension 2 and higher, the total and communication times for GS1R are less. The GS1R communication times are comparable to the commf times of the CLW-CT2. The commp times are greater than the commf times because the message lengths are higher. The differences increase with dimension. Tables 7(c) and 8(c) show similar trends illustrated in Tables 7(b) and 8(b). The timings of the inverse transform CLW-GS1 are similar to those of the forward transform.

To conclude the experimental comparison of the CLW and compact transforms for real sequences, the computational times of the two are about the same. The differences in the dimension 0 timings are not understood, however. The index switch time makes up a nontrivial percentage of the total time for low dimensions and decreases the viability of the compact transforms. The local movement of data could be speeded up considerably by efficient use of cache. The communication times for the compact

TABLE 8(a)
Averaged milliseconds per FFT *of a real sequence (forward transform) and a conjugate-symmetric sequence (inverse transform). The Cooley-Lewis-Welch algorithm is used. Sequence length*: $N = 4096$; *number of sequences*: $q = 1$.

	CLW-CT2 forward transform						CLW-GS1 inverse transform					
Dim	Total	Calcf	Calcp	Commf	Commp	Eff	Total	Calcf	Calcp	Commf	Commp	Eff
0	177.15	158.59	18.56	0.00	0.00	1.00	175.36	157.82	17.28	0.00	0.00	1.00
1	106.05	79.42	15.23	5.63	5.44	0.84	104.51	78.34	14.85	5.63	5.38	0.84
2	59.20	40.74	7.68	4.54	5.66	0.75	58.24	40.06	7.46	4.67	5.60	0.75
3	33.57	21.09	3.89	3.38	4.53	0.66	32.88	20.43	3.79	3.44	4.54	0.67
4	19.78	10.95	2.00	2.62	3.31	0.56	19.46	10.66	1.98	2.69	3.38	0.56
5	12.73	5.63	1.00	2.19	2.68	0.43	12.55	5.50	1.02	2.34	2.67	0.43
6	9.11	3.06	0.57	2.19	2.26	0.30	9.02	2.94	0.56	2.22	2.26	0.30

TABLE 8(b)
Averaged milliseconds per FFT *of a real sequence (forward transform) and a conjugate-symmetric sequence (inverse transform). The Cooley-Lewis-Welch algorithm is used. Sequence length*: $N = 4096$; *number of sequences*: $q = 8$.

	CLW-CT2 forward transform						CLW-GS1 inverse transform					
Dim	Total	Calcf	Calcp	Commf	Commp	Eff	Total	Calcf	Calcp	Commf	Commp	Eff
0	1003.52	889.98	113.41	0.00	0.00	1.00	980.61	877.70	102.91	0.00	0.00	1.00
2	344.67	224.38	46.08	31.87	42.02	0.73	340.86	221.70	44.22	32.32	42.02	0.72
3	190.18	113.02	23.01	21.74	31.79	0.67	188.34	111.74	22.16	21.95	31.81	0.65
4	104.93	56.98	11.55	14.08	21.58	0.60	104.22	56.41	11.14	14.30	21.62	0.59
5	58.68	28.81	5.86	9.10	14.03	0.53	58.42	28.56	5.60	9.28	14.06	0.52
6	33.66	14.57	2.95	6.01	9.00	0.47	33.46	14.36	2.83	6.19	8.97	0.46

TABLE 8(c)
Averaged milliseconds per FFT *of a real sequence (forward transform) and a conjugate-symmetric sequence (inverse transform). The Cooley-Lewis-Welch algorithm is used. Sequence length*: $N = 4096$; *number of sequences*: $q = 16$.

	CLW-CT2 forward transform						CLW-GS1 inverse transform					
Dim	Total	Calcf	Calcp	Commf	Commp	Eff	Total	Calcf	Calcp	Commf	Commp	Eff
0	1947.78	1725.95	221.82	0.00	0.00	1.00	1907.33	1706.50	200.70	0.00	0.00	1.00
3	368.90	217.90	44.85	42.38	63.06	0.66	366.10	216.50	43.12	42.66	63.09	0.65
4	202.41	109.60	22.58	27.06	42.38	0.60	201.29	109.04	21.60	27.35	42.42	0.59
5	111.22	55.15	11.29	16.86	26.99	0.55	110.90	54.97	10.86	17.16	26.94	0.54
6	61.96	27.81	5.66	10.68	16.74	0.49	61.88	27.73	5.48	10.92	16.70	0.48

transforms are about half those of the CLW scheme. When the communication time becomes significant, i.e., at higher dimensions, the compact transforms offer a time savings over CLW.

7. Summary. We have presented eight in-place, compact FFTs of real sequences for distributed-memory multiprocessors. i-cycles are used for communication and result in minimal communication costs on hypercubes. While the algorithms are written for

MIMD processors, by suitable redefinition of operations, they can be implemented efficiently on SIMD processors also. If the real sequence is naturally ordered, then the transform requires $d+2$ i-cycles, where d is the dimension. If the conjugate-symmetric sequence is in conditional ordering, which is nearly natural ordering, then only $d+1$ i-cycles are required. The communication time is comparable to the complex FFT due to Swarztrauber [13] and Walker [14], whereas the computational complexity is comparable to the parallel CLW [4] algorithm.

The parallel overhead for all the schemes scales as $\beta\, d/m$ in the large granularity limit, where β is the message-passing rate, d is the processor dimension, and m is the base 2 log of the sequence length.

We believe that the conditional ordering is the key to efficient algorithms for cosine, sine, and quarter-wave transform.

A comparison of the compact and CLW algorithms as implemented on an nCUBE/2 multiprocessor shows that while computational times are similar, the communication times are about half in the compact algorithms. Local memory access time must be increased, however, for the compact schemes to compete at low dimensions with the CLW scheme.

Acknowledgments. I would like to thank Roland Sweet at the University of Colorado,Denver, Paul Swarztrauber at the National Centre for Atmospheric Research, and David Walker at the Oak Ridge National Laboratory for helpful suggestions. Timings were run at the CAIP Parallel Processing Laboratory. CAIP, Center for Computer Aids for Industrial Productivity, is an Advanced Technology Center of the New Jersey Commission of Science and Technology at Rutgers University.

REFERENCES

[1] G. D. Bergland, *A fast Fourier transform for real-valued series*, Comm. ACM, 11 (1968), pp. 703-710.

[2] W. L. Briggs, *Further symmetries of in-place* FFTs, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 644-654.

[3] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, Berlin, 1987.

[4] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, *The fast Fourier transform algorithm: programming considerations in the calculation of sine, cosine and Laplace transforms*, J. Sound Vib., 12 (1970), pp. 315-337.

[5] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.

[6] D. Fraser, *Array permutation by index-digit permutation*, J. ACM, 22 (1976), pp. 298-308.

[7] W. M. Gentleman and G. Sande, *Fast Fourier transforms for fun and profit*, in Proc. AFIPS Joint Computer Conference, 29 (1966), pp. 563-578.

[8] R. B. Pelz, *The parallel Fourier pseudospectral method*, J. Comput. Phys., 92 (1991), pp. 296-312.

[9] ———, *Parallel Fourier spectral methods on ensemble architectures*, Comput. Meth. Appl. Mech. Engrg., 89 (1991), pp. 529-542.

[10] ———, *The parallel Chebyshev pseudospectral method: Cosine transform and derivative recursion*, in Proc. Fourth Conf. on Hypercubes, Concurrent Computers, and Applications, 1989, pp. 433-439.

[11] J. Salmon, *An astrophysical N-body simulation for hypercubes*, Tech. Rep. HM-78, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA, 1984.

[12] P. N. Swarztrauber, *Symmetric* FFTs, Math. Comp., 47 (1986), pp. 323-346.

[13] ———, *Multiprocessor* FFTs, Parallel Comput., 5 (1987), pp. 197-210.

[14] D. W. Walker, *Portable programming within a message-passing model: The* FFT *as an example*, in Proc. Third Conference on Hypercube Concurrent Computers and Applications, G. C. Fox, ed., ACM Press, 1988, pp. 1438-1450.

[15] P. H. Worley and D. W. Walker, *Parallelizing the spectral transform method—Part* I, TM-11747, Oak Ridge National Laboratory, 1991; Concurrency: Practice and Experience, to appear.

[16] D. W. WALKER, P. H. WORLEY, AND J. B. DRAKE, *Parallelizing the spectral transform method—Part* II, TM-11855, Oak Ridge National Laboratory, 1991; Concurrency: Practice and Experience, submitted.

[17] Partial results were presented as a contributed paper at the SIAM Conf. on Parallel Processing for Scientific Computing, Houston, TX, March 25-27, 1991. Similar algorithms are being developed by R. Sweet, private communication.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 936-952, July 1993

THE DEVELOPMENT OF VARIABLE-STEP SYMPLECTIC INTEGRATORS, WITH APPLICATION TO THE TWO-BODY PROBLEM*

M. P. CALVO† AND J. M. SANZ-SERNA†

Abstract. The authors develop and test variable step symplectic Runge-Kutta-Nyström algorithms for the integration of Hamiltonian systems of ordinary differential equations. Numerical experiments suggest that, for symplectic formulae, moving from constant to variable stepsizes results in a marked decrease in efficiency. On the other hand, symplectic formulae with constant stepsizes may outperform available standard (nonsymplectic) variable-step codes. For the model situation consisting in the long-time integration of the two-body problem, our experimental findings are backed by theoretical analysis.

Key words. symplectic integration, Kepler's problem, Runge-Kutta-Nyström methods

AMS subject classifications. 65L05, 70H05, 70F05

1. Introduction.

In mechanics, optics, chemistry, etc., situations where dissipation does not play a significant role may be modelled by means of Hamiltonian systems of ordinary differential equations (ODEs) or partial differential equations (PDEs) [2]. Hamiltonian systems of ODEs are of the form

$$\dot{p}^I = -\partial H/\partial q^I, \quad \dot{q}^I = \partial H/\partial p^I, \quad 1 \leqq I \leqq d, \tag{1.1}$$

where the integer d is the number of degrees of freedom, the Hamiltonian $H = H(\mathbf{p}, \mathbf{q}) = H(p^1, \dots, p^d, q^1, \dots, q^d)$ is a sufficiently smooth, real function of $2d$ real variables, and a dot represents differentiation with respect to t (time). There has been much recent interest in the numerical integration of (1.1) by means of so-called symplectic or canonical integrators, starting with the work of Ruth [12], Feng [7], and Channell and Scovel [4]. An extensive list of references can be found in the survey [14].

In order to explain in simple terms the meaning and relevance of symplecticness, it is advisable to consider first the question of how to tell, from the knowledge of the *solutions* of a system of ODEs, whether the *system* is of Hamiltonian form or otherwise. More precisely, let $\mathscr{S}$ be an autonomous system of ODEs for the dependent variables $(\mathbf{p}, \mathbf{q})$, and let us introduce the $\mathbb{R}^{2d}$-valued function $\varphi_t(\mathbf{p}_0, \mathbf{p}_0)$ such that, for fixed $\mathbf{p}_0$ and $\mathbf{q}_0$ and varying t, $(\mathbf{p}(t), \mathbf{q}(t)) = \varphi_t(\mathbf{p}_0, \mathbf{q}_0)$ is the solution of $\mathscr{S}$ with initial condition $\mathbf{p}(0) = \mathbf{p}_0$, $\mathbf{q}(0) = \mathbf{q}_0$. If we now see t as a parameter and $\mathbf{p}_0, \mathbf{q}_0$ as variables, $\varphi_t(\mathbf{p}_0, \mathbf{q}_0)$ defines a transformation in the space $\mathbb{R}^{2d}$ (the phase space). This transformation is the *flow* of the differential system $\mathscr{S}$. If we were given φ_t and at the same time $\mathscr{S}$ were concealed from us, could we tell whether $\mathscr{S}$ is a Hamiltonian system or otherwise? The answer to this question is affirmative. The system $\mathscr{S}$ is Hamiltonian *if and only if, for each* t, φ_t *is a symplectic transformation.* Now a transformation $\mathscr{T}$ in phase space is said to be symplectic [2] if for any bounded two-dimensional surface D in phase space, the sum of the two-dimensional (signed) areas of the d projections of D onto the planes (p^I, q^I) is the same as the sum of the two-dimensional (signed) areas of the d projections of $\mathscr{T}(D)$ onto the planes (p^I, q^I). Thus the symplectic character of

* Received by the editors December 18, 1991; accepted for publication (in revised form) April 30, 1992. This research was supported by Junta de Castilla y León under project 1031-89 and by Dirección General de Investigación Científica y Técnica under project PB89-0351.

† Departamento de Matemática Aplicada y Computación, Facultad de Ciencias, Universidad de Valladolid, Valladolid, Spain (Maripaz@cpd.uva.es and sanzserna@cpd.uva.es).

the flow is the hallmark of Hamiltonian systems. Hamiltonian problems have many specific features not shared by other systems of differential equations. All such specific features (absence of attractors, recurrence, etc.) directly derive from the symplecticness of the corresponding flow [2].

A one-step numerical method used with steplength h defines a transformation in phase space $\psi_h(\mathbf{p}_0, \mathbf{q}_0)$ that advances the solution h units of time, starting from $(\mathbf{p}_0, \mathbf{q}_0)$. Of course, $\psi_h(\mathbf{p}_0, \mathbf{q}_0)$ is an approximation to $\varphi_h(\mathbf{p}_0, \mathbf{q}_0)$, and the numerical method approximates $\varphi_{nh} = \varphi_h^n$ by iterating n times ψ_h. For Hamiltonian problems integrated by classical methods, such as explicit Runge-Kutta methods, the transformation ψ_h turns out to be *nonsymplectic.* Then the numerical method misses the important specific features associated with symplectic transformations. However, there are *symplectic* methods for which ψ_h is guaranteed to be symplectic for Hamiltonian problems.

Numerical experiments have shown that for Hamiltonian problems, symplectic integrators may well be an improvement over their nonsymplectic counterparts. However, the development of symplectic methods has so far been confined to *constant stepsize* formulae and, accordingly, numerical tests have used as reference algorithms constant stepsize implementations of classical methods. Such implementations are, by modern numerical ODE standards, very naive, and the question arises of whether, for Hamiltonian problems, a symplectic method with constant stepsizes, may actually be more efficient than a modern variable-step code. Before we carried out the experiments reported in this paper, we felt that the answer to that question would be no. On the other hand, we suspected that for Hamiltonian problems, variable stepsize symplectic algorithms would improve on standard variable stepsize algorithms. Accordingly, we decided to develop variable stepsize symplectic algorithms.

In this paper we report on our experience with the construction and assessment of *variable-step, symplectic, explicit Runge-Kutta-Nyström algorithms.* We used Runge-Kutta-Nyström (RKN) methods rather than Runge-Kutta methods because all symplectic Runge-Kutta formulae are implicit [13]. It appears that both our guesses above were wrong: *constant stepsize symplectic methods may beat standard variable stepsize codes, but variable stepsize symplectic codes are not more advantageous than standard variable stepsize codes.*

Section 2 is devoted to the construction of the symplectic RKN code. The results of the numerical experiments are presented in § 3, where we use as a test problem the well-known two-body (Kepler) problem. In § 4 we analyze our experimental findings. In particular, we provide a complete theoretical study of the performance of general one-step numerical methods in the integration of the two-body problem. Finally, in § 5, we present our conclusions.

2. Construction of a symplectic RKN code.

2.1. RKN methods. We restrict our attention to systems of the special form

$$\dot{\mathbf{p}} = \mathbf{f}(\mathbf{q}), \qquad \dot{\mathbf{q}} = \mathbf{p} \tag{2.1}$$

(i.e., to second-order systems $\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q})$). If $\mathbf{f}$ is the gradient of a scalar function $-V(\mathbf{q})$, then (2.1) is a Hamiltonian system with

$$H = H(\mathbf{p}, \mathbf{q}) = T(\mathbf{p}) + V(\mathbf{q}), \qquad T(\mathbf{p}) = \tfrac{1}{2}\mathbf{p}^T\mathbf{p}.$$

In mechanics, the $\mathbf{q}$ variables represent Lagrangian coordinates, the $\mathbf{p}$ variables the corresponding momenta, $\mathbf{f}$ the forces, T is the kinetic energy, V the potential energy, and H the total energy [2].

An explicit RKN method for (2.1) takes the form [5], [8]

$$
\begin{aligned}
\mathbf{Q}_i &= \mathbf{q}_n + h\gamma_i \mathbf{p}_n + h^2 \sum_{j<i} \alpha_{ij}\mathbf{f}(\mathbf{Q}_j),\\
\mathbf{p}_{n+1} &= \mathbf{p}_n + h \sum_{i=1}^{s} b_i \mathbf{f}(\mathbf{Q}_i),\\
\mathbf{q}_{n+1} &= \mathbf{q}_n + h\mathbf{p}_n + h^2 \sum_{i=1}^{s} \beta_i \mathbf{f}(\mathbf{Q}_i),
\end{aligned}
\tag{2.2}
$$

where we assume, unless otherwise stated, that the following well-known condition [5], [8] holds:

$$\beta_i = b_i(1-\gamma_i), \qquad 1 \leqq i \leqq s. \tag{2.3}$$

As in [5], we consider first same as last (FSAL) methods, i.e., methods with

$$\gamma_1 = 0, \qquad \gamma_s = 1, \tag{2.4a}$$

$$\alpha_{sj} = \beta_j, \qquad 1 \leqq j \leqq s-1. \tag{2.4b}$$

Note that (2.4a) implies, via (2.3), that $\beta_s = 0$, and then the last stage $\mathbf{Q}_s$ of the current step coincides with $\mathbf{q}_{n+1}$, which, in turn, is the first stage of the next step. Therefore, a step of an FSAL s-stage method requires only $s-1$ evaluations of $\mathbf{f}$.

The method (2.2) is symplectic if [21], [11], [3], [14]

$$\alpha_{ij} = b_j(\gamma_i - \gamma_j), \qquad i > j. \tag{2.5}$$

For symplectic methods with s stages, we have s coefficients b_i and s coefficients γ_i as free parameters; the coefficients β_i and α_{ij} are determined by (2.3) and (2.5), respectively. On the other hand (2.3), (2.4a), and (2.5) imply (2.4b), so that a symplectic FSAL method has s coefficients b_i and $s-2$ coefficients $\gamma_i, 2 \leqq i \leqq s-1$, as free parameters.

2.2. Derivation of a fourth-order, symplectic, FSAL RKN method. To construct a variable-step symplectic code, we decided to begin with a fourth-order formula. While higher-order formulae are expected to be more efficient, they are also more difficult to construct. For a method (2.2)-(2.3) to have order four, the coefficients should satisfy *seven* order conditions [8]. However, for symplectic methods, not all order conditions are independent [1], [3], [14], [15] and, in fact, it turns out [3] that it is sufficient to impose only *six* of them. For FSAL symplectic methods, four stages furnish six free coefficients, and after imposing order four, no room is left for "tuning" the formula. We then settle for five-stage FSAL symplectic methods, for which a two-parameter family of order-four methods exists. Following a standard practice (see [5] and [6]) we choose among the members of this family the method with "smallest" truncation error.

For smooth problems the $\mathbf{p}$-truncation and $\mathbf{q}$-truncation errors of an RKN method, respectively, possess Taylor expansions of the form [5], [8]

$$\sum_{i=0}^{\infty} h^{i+1} \sum_j c_j'^{(i+1)} \mathbf{F}_j^{(i+1)} \tag{2.6a}$$

and

$$\sum_{i=1}^{\infty} h^{i+1} \sum_k c_k^{(i+1)} \mathbf{F}_k^{(i)}, \tag{2.6b}$$

where the $\mathbf{F}_j^{(i)}$ are the elementary differentials that only depend on the problem (2.1) being integrated, and the $c_j'^{(i+1)}$ and $c_k^{(i+1)}$ are polynomials in the method coefficients α_{ij}, γ_i, β_i, b_i. In (2.6a), the sum in j is extended to all special Nyström trees with $i+1$ nodes, while in (2.6b), the sum in k is extended to all special Nyström trees with i nodes. For fourth-order methods, $c_j'^{(i)}$ and $c_k^{(i)}$ vanish for $i \leqq 4$ and we try to minimize $c_j'^{(5)}$ and $c_k^{(5)}$. We proceed as follows. Let us denote by $\mathbf{c}'^{(i)}$ and $\mathbf{c}^{(i)}$, respectively, the vectors with components $c_j'^{(i)}$ and $c_k^{(i)}$, and set

$$A'^{(5)} = \|\mathbf{c}'^{(5)}\|, \qquad A^{(5)} = \|\mathbf{c}^{(5)}\|. \tag{2.7}$$

(The norm is the standard Euclidean norm.) We consider $\phi = (A'^{(5)})^2 + (A^{(5)})^2$ as a function of the eight free coefficients γ_i, $2 \leqq i \leqq 4$, and b_j, $1 \leqq j \leqq 5$, and use the NAG subroutine E04UCF to minimize ϕ subject to the six equality constraints that impose order four and subject to bounds $-1.5 \leqq \gamma_i, b_j \leqq 1.5$. Of course, the minimization subroutine requires an initial guess for the minimum and converges only to a local minimum that depends on the initial guess. A thousand random initial guesses (subject to $-1.5 \leqq \gamma_i, b_j \leqq 1.5$) were taken, and we kept the local minimum with the smallest value of ϕ. The method thus obtained does not satisfy to machine precision the conditions for order four, because the NAG routine fails in exactly enforcing the equality constraints. We then kept the values b_1 and b_2 provided by the minimization routine and determined γ_i, $2 \leqq i \leqq 4$, and b_j, $3 \leqq j \leqq 5$, by solving the six equations for order four by means of Newton's method in quadruple precision. This of course resulted in a solution that, while being close to that provided by the minimization procedure, satisfies the order conditions to a very high precision. The coefficients are given by

$$\begin{aligned} &\gamma_1 = 0, && b_1 = 0.061758858135626325,\\ &\gamma_2 = 0.205177661542286386, && b_2 = 0.338978026553643355,\\ &\gamma_3 = 0.608198943146500973, && b_3 = 0.614791307175577566,\\ &\gamma_4 = 0.487278066807586965, && b_4 = -0.140548014659373380,\\ &\gamma_5 = 1, && b_5 = 0.125019822794526133, \end{aligned} \tag{2.8}$$

along with (2.3) and (2.5).

For this method the quantities in (2.7) are $A'^{(5)} = 0.00067$ and $A^{(5)} = 0.00071$. As a reference method for the numerical tests, we employ the fourth-order, FSAL, nonsymplectic formula of Dormand, El-Mikkawy, and Prince [5, Table 3]. This has four stages (three evaluations) and $A'^{(5)} = 0.0018$, $A^{(5)} = 0.00046$. Thus, per step, the reference method achieves an accuracy comparable to that of the symplectic method (2.8), but is cheaper by a factor of 3/4. In general, symplectic integrators require, for the same accuracy, more work than their nonsymplectic counterparts since, to impose symplecticness, free parameters are sacrificed that could otherwise be directed at achieving accuracy.

2.3. Error estimation. The standard way [5], [8] of estimating the errors in a pth-order RKN method (2.2) is to supplement (2.2) with formulae

$$\begin{aligned} \hat{\mathbf{p}}_{n+1}\mathbf{p}_n + h \sum_{i=1}^{s} \hat{b}_i \mathbf{f}(\mathbf{Q}_i),\\ \hat{\mathbf{q}}_{n+1} = \mathbf{q}_n + h\mathbf{p}_n + h^2 \sum_{i=1}^{s} \hat{\beta}_i \mathbf{f}(\mathbf{Q}_i) \end{aligned} \tag{2.9}$$

in such a way that $(\mathbf{p}_n, \mathbf{q}_n) \mapsto (\hat{\mathbf{p}}_{n+1}, \hat{\mathbf{q}}_{n+1})$ is an RKN method of order $q<p$ (usually $q=p-1$ or $q=p-2$). Of course, the computation of $(\hat{\mathbf{p}}_{n+1}, \hat{\mathbf{q}}_{n+1})$ employs *the same* function evaluations $\mathbf{f}(\mathbf{Q}_i)$ that are used to compute $(\mathbf{p}_{n+1}, \mathbf{q}_{n+1})$. The difference between the low-order $(\hat{\mathbf{p}}_{n+1}, \hat{\mathbf{q}}_{n+1})$ and high-order $(\mathbf{p}_{n+1}, \mathbf{q}_{n+1})$ results is then taken to be an approximation to the local error at the step $n \mapsto n+1$. For (2.8), we take the order q of the embedded method to be 3.

The weights $\hat{b}_i$, $1 \leqq i \leqq 5$, must satisfy four equations for the local error in $\hat{\mathbf{p}}_{n+1}$ to be $O(h^4)$. These equations are linear in the $\hat{b}_i$'s and it is a simple matter to express $\hat{b}_i$, $1 \leqq i \leqq 4$, in terms of $\hat{b}_5$, which remains a free parameter. The value of $\hat{b}_5$ is chosen according to a procedure suggested by Dormand and Prince. The quantities

$$C'^{(5)} = \frac{\|\mathbf{c}'^{(5)} - \hat{\mathbf{c}}'^{(5)}\|}{\|\hat{\mathbf{c}}'^{(4)}\|}, \qquad B'^{(5)} = \frac{\|\hat{\mathbf{c}}'^{(5)}\|}{\|\hat{\mathbf{c}}'^{(4)}\|} \tag{2.10}$$

should be made as small as possible (letters with a hat refer, of course, to the lower-order method). The Taylor expansion of the $\mathbf{p}$-component of the error estimator $\mathbf{p}_{n+1} - \hat{\mathbf{p}}_{n+1}$ has coefficients $-\hat{c}_j'^{(4)}$ in the order $O(h^4)$ terms and coefficients $c_j'^{(5)} - \hat{c}_j'^{(5)}$ in the order $O(h^5)$ terms (cf. (2.6a)). Thus, a small $C'^{(5)}$ ensures that, in the ($\mathbf{p}$-component of) the error estimator, the leading $O(h^4)$ term dominates over the next, $O(h^5)$, term of the Taylor expansion. This is beneficial, since the mechanism for stepsize selection assumes an $O(h^4)$ behaviour in the estimator. On the other hand, a small $B'^{(5)}$ ensures that the third-order formula used for estimation is sufficiently different from the fourth-order formula used for timestepping. (Note that as the third-order formula comes closer to the fourth-order formula, the denominator in $B'^{(5)}$ tends to 0 and hence $B'^{(5)}$ tends to infinity.) We minimize the function $\phi(\hat{b}_5) = (B'^{(5)})^2 + (C'^{(5)})^2$ by the simple procedure of evaluating ϕ at uniformly spaced values of $\hat{b}_5$ (the spacing used was 0.01). This yields $\hat{b}_5 = 0.2$.

The coefficients $\hat{\beta}_i$, $1 \leqq i \leqq 5$, are seen as free parameters, i.e., they are not derived from $\hat{b}_i$ through (2.3). For the local error in $\hat{\mathbf{q}}_{n+1}$ to be $O(h^4)$, the $\hat{\beta}_i$, $1 \leqq i \leqq 5$, must satisfy two (linear) equations; this leaves three free parameters. We arbitrarily set $\hat{\beta}_5 = 0$ and expressed $\hat{\beta}_1$ and $\hat{\beta}_2$ in terms of $\hat{\beta}_3$ and $\hat{\beta}_4$. The free $\hat{\beta}_3, \hat{\beta}_4$ are now chosen to minimize $(B^{(5)})^2 + (C^{(5)})^2$, where

$$C^{(5)} = \frac{\|\mathbf{c}^{(5)} - \hat{\mathbf{c}}^{(5)}\|}{\|\hat{\mathbf{c}}^{(4)}\|}, \qquad B^{(5)} = \frac{\|\hat{\mathbf{c}}^{(5)}\|}{\|\hat{\mathbf{c}}^{(4)}\|}. \tag{2.11}$$

The minimization was again performed by sampling the objective function on a grid with 0.01×0.01 spacing. The weights of the third-order formula (2.9) embedded in (2.5) are as follows:

$$\begin{aligned}
&\hat{b}_1 = -0.127115143890665440, &&\hat{\beta}_1 = 0.110014238746029571,\\
&\hat{b}_2 = 0.698831995430764851, &&\hat{\beta}_2 = 0.189985761253970428,\\
&\hat{b}_3 = 0.375269477646788521, &&\hat{\beta}_3 = 0.25,\\
&\hat{b}_4 = -0.146986329186887931, &&\hat{\beta}_4 = -0.05,\\
&\hat{b}_5 = 0.2, &&\hat{\beta}_5 = 0.
\end{aligned} \tag{2.12}$$

With this choice the quantities in (2.10) and (2.11) are

$$C'^{(5)} = 1.06, \quad B'^{(5)} = 1.06, \quad C^{(5)} = 0.47, \quad B^{(5)} = 0.25.$$

For the fourth-order nonsymplectic scheme used as a reference method, Dormand, El-Mikkawy, and Prince [5] provide an embedded formula with

$$C'^{(5)} = 1.19, \quad B'^{(5)} = 1.20, \quad C^{(5)} = 1.02, \quad B^{(5)} = 1.03.$$

This shows that the minimizations we carried out above are as successful as those in [5].

2.4. Implementation. The embedded pair (2.8), (2.12) and the reference-embedded pair of Dormand, El-Mikkawy, and Prince were implemented in a standard way following very closely the code DOPRIN in [8].

3. Numerical results. Several test problems, including integrable and nonintegrable Hamiltonians, were used. The main conclusions as to the relative merit of the various algorithms do not greatly depend on the particular test problem, and hence we only report on the results corresponding to the Newton potential [2] $V(q^1, q^2) = -1/\|\mathbf{q}\|$ with initial condition

$$p^1 = 0, \quad p^2 = \sqrt{\frac{1+e}{1-e}}, \quad q^1 = 1-e, \quad q^2 = 0.$$

Here e is a parameter $0 \leqq e < 1$. The solution is 2π-periodic and its projection onto the (configuration) $\mathbf{q}$-space is an ellipse with eccentricity e and major semiaxis 1. Initially, the moving mass is at the pericentre of the ellipse (i.e., the closest it can be to the coordinate origin). After half a period (apocentre), its distance r to the origin is $1+e$. Thus $r_{\max}/r_{\min} = (1+e)/(1-e)$, which is large for large eccentricities. Moreover, the ith derivatives of the force $\mathbf{f}$ behave like $r^{-(i+2)}$, so that, for large eccentricities, the elementary differentials of high order may vary by several orders of magnitude along the orbit. In fact, this well-known test problem with large e (say, $e = 0.9$) is often taken as a "severe test for the stepsize control procedure" of ODE algorithms [6].

The test problem was integrated by combining each of the eccentricities 0.1, 0.3, 0.5, 0.7, and 0.9 with each of the final times $10 \times 2\pi$, $30 \times 2\pi$, $90 \times 2\pi$, $270 \times 2\pi$, $810 \times 2\pi$, $2430 \times 2\pi$, $7290 \times 2\pi$, and $21870 \times 2\pi$. We were particularly interested in long time intervals, as it is in this sort of simulation that the advantages of symplecticness should be felt (see [14]). For short time intervals, the local error of the formula is of paramount importance, and it is as the time interval gets larger that advantages derived from a better qualitative behaviour become more prominent. In celestial mechanics very long time integrations are often required with potentials that are small perturbations of the two-body potential considered here.

In the tests we used the symplectic variable-step code (SV), the nonsymplectic variable-step code (NSV), and also fixed-step implementations of the symplectic formulae (SF) and nonsymplectic formulae (NSF). The variable-step codes were tried with absolute error tolerances of $10^{-4}, 10^{-5}, \ldots, 10^{-11}$, and the fixed-step algorithms were run with stepsizes $2\pi/16, 2\pi/32, \ldots, 2\pi/2048$. Errors were measured in the Euclidean norm of $\mathbb{R}^4$.

Figure 1 gives, for $e = 0.5$ and a final time of 21870 periods, the final error against the computational effort measured by the number of $\mathbf{f}$-evaluations. The figure contains information for the runs that yielded errors in the 10^{-1} to 10^{-4} range, namely,

(i) SV with tolerances 10^{-10}, 10^{-11} (plus signs joined by a dashed line);

(ii) NSV with tolerances 10^{-9}, 10^{-10}, 10^{-11} (circles joined by a solid line);

(iii) SF with timestep $2\pi/256$, $2\pi/512$, $2\pi/1024$ (stars joined by a dashed-dotted line);

(iv) NSF with timestep $2\pi/2048$ ($a \times$ sign).

Let us first compare the results of SF and NSF. Recall that these RKN formulae have error constants of roughly the same size, but SF has four evaluations per step against three evaluations per step in NSF. Thus, on *local error considerations alone*,

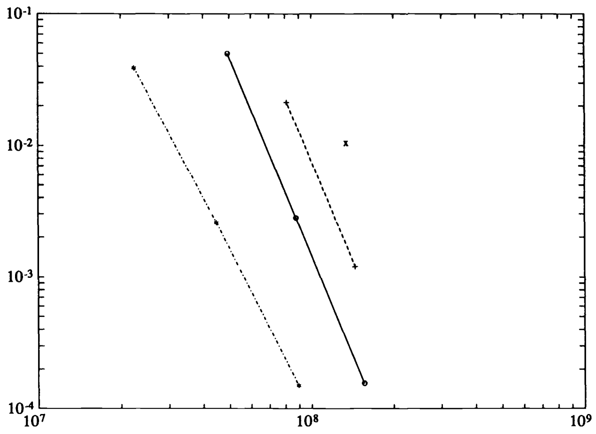


FIG 1. *Error against number of function evaluations, after* 21870 *periods,* $e = 0.5$.

we would expect that for the same global error, the numbers of evaluations of the NSF and SF would be in a ratio 3/4. On the contrary, the experimental results show that, for the same error, the symplectic formula is *four times* less expensive than the nonsymplectic process (ratio 4/1). This shows that there is something in the error propagation mechanism of the symplectic algorithm that gives it a clear advantage over its nonsymplectic counterpart. In the next section we prove rigorously that in the asymptotic expansion of the global error of the symplectic formula, the coefficient of the powers h^4, h^5, h^6, and h^7 grows linearly with the integration time t. Thus in the symplectic formula, for small errors, we need h to be small with respect to $t^{-1/4}$. On the other hand, for the nonsymplectic formula, the coefficient of the leading h^4 term of the global error also increases linearly with t, but the h^5, h^6, and h^7 terms possess coefficients that grow like t^2. If t is large, for small errors, h should be small with respect to $t^{-2/5}$. This is to be compared with $h \ll t^{-1/4}$ for the symplectic case. This shows that for large t symplecticness pays. In fact, when $e = 0.5$, SF improves on NSF if t_{final} is larger than, say, 30 periods.

Turning now to a comparison between NSF and NSV, we observe that for the formula of Dormand, El-Mikkawy, and Prince the use of variable-stepsizes results in a gain in efficiency by a factor of two. In the apocentre, the variable-stepsize code takes stepsizes about seven times as large as those it takes near the pericentre, with the result that, as expected, NSV saves on function evaluations for a given error. Note that the line which joins the NSV points has slope -5 in spite of the method having order four. This is again due to the fact that the coefficient in the leading h^4 term in the global error grows linearly with t, while the coefficients of the subsequent terms grow like t^2; for t large, $t^2h^5 \gg th^4$ and the method behaves as if its order were five (see § 3).

On the other hand, for the symplectic formula, going from fixed to variable-stepsizes results in a *decrease* in efficiency. We will return to this point later. For the present, let us note that, with variable stepsizes, the line joining the points of the symplectic algorithm are in agreement with fifth-order behaviour of the error. In fact, SV and NSV show very similar behaviour. The only difference between them lies in the fact that, for a given error, the costs of NSV and SV are in a ratio 3/4, i.e., in the ratio we would have anticipated from a study of the local errors without taking symplecticness into account.

Like Fig. 1, Fig. 2 corresponds to a final time $21870 \times 2\pi$, but now $e = 0.3$. Again we have displayed the results corresponding to runs for which the errors lie in the 10^{-1} to 10^{-4} range. These are the following:

(i) SV with tolerances 10^{-10}, 10^{-11} (plus signs joined by a dashed line);

(ii) NSV with tolerances 10^{-9}, 10^{-10} (circles joined by a solid line);

(iii) SF with timestep $2\pi/128$, $2\pi/256$, $2\pi/512$ (stars joined by a dashed-dotted line);

(iv) NSF with timestep $2\pi/1024$, $2\pi/2048$ ($\times$ sign, dotted line).

We see that the overall pattern is not changed by changing the eccentricity. The NSV and SV algorithms have efficiencies that are still in the predicted 3/4 ratio. On the other hand, with $e = 0.3$, the advantages of NSV over NSF are less marked, as we would have expected. In fact, for $e = 0.3$, both variable-step codes only vary the stepsize along the orbit by a factor of 3. The NSF points, which for $e = 0.5$ were to the right of the SV dashed line, are now exactly on this SV line.

Figure 3 corresponds to the same final time with $e = 0.7$. The following runs are represented (results for NSF are not reported for stepsizes used, as errors below 10^{-1} could not be obtained):

(i) SV with tolerances 10^{-10}, 10^{-4} (plus signs joined by a dashed line);

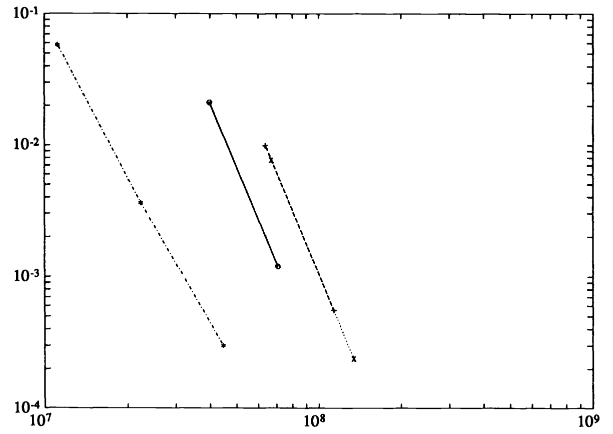


FIG. 2. *Error against number of function evaluations, after* 21870 *periods,* $e = 0.3$.

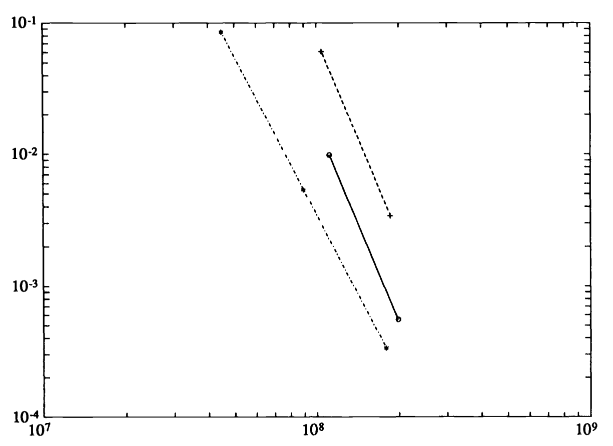


FIG. 3. *Error against number of function evaluations, after* 21870 *periods,* $e = 0.7$.

(ii) NSV with tolerances 10^{-10}, 10^{-11} (circles joined by a solid line);

(iii) SF with time step $2\pi/512, 2\pi/1024, 2\pi/2048$ (stars joined by a dashed-dotted line).

Now SV and NSV become more efficient and change h by a factor of 22. Nevertheless, SF is still the most efficient method: the advantages of symplecticness are not offset by the disadvantages of constant h.

For the smaller values of t_{final} that we tried, the picture is very much the same, except if t_{final} is not large and e is large, SF is the most efficient method, NSV is second, and SV is 4/3 times worse than NSV. For fixed t_{final}, as e approaches 1, the benefits of variable steps become more prominent and NSV improves on SF. For fixed e, as t_{final} increases, the benefits of symplecticness dominate and SF improves on NSV.

Figure 4 gives, for $e = 0.5$, error against time for SV (tolerance 10^{-10}), NSV (tolerance 10^{-9}), SF ($h = 2\pi/1024$), and NSF ($h = 2\pi/2048$). For SF the error shows a linear behaviour with respect to t, as stated earlier. For the other methods the error grows like t^2.

Figure 5 displays, for $e = 0.5$, the error in energy $|H(\mathbf{p}_n, \mathbf{q}_n) - H(\mathbf{p}(t_n), \mathbf{q}(t_n))|$ against time. Of course, the theoretical solution preserves energy $H(\mathbf{p}(t), \mathbf{q}(t)) = H(\mathbf{p}(0), \mathbf{q}(0))$ and consequently the error in energy equals the energy growth

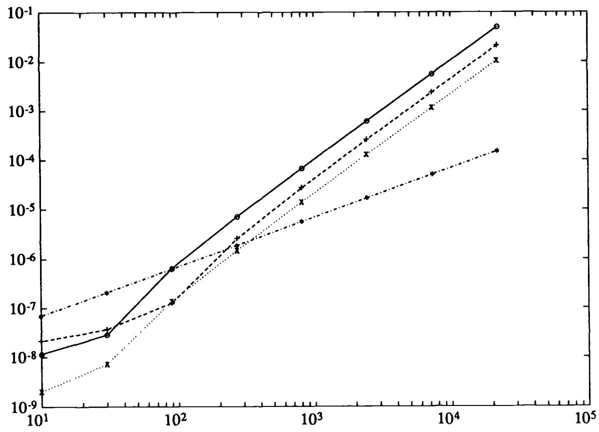


FIG. 4. *Error against time in periods*, $e = 0.5$.

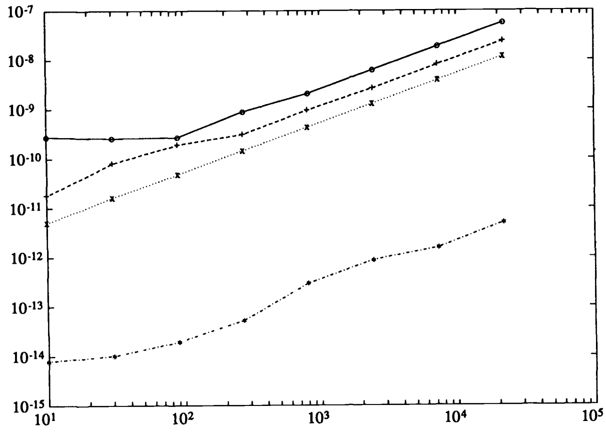


FIG. 5. *Error in energy against time in periods*, $e = 0.5$.

$|H(\mathbf{p}_n, \mathbf{q}_n) - H(\mathbf{p}(0), \mathbf{q}(0))|$. The runs displayed are similar to those shown in Fig. 4. In SV, NSV, and NF the error in energy grows linearly with t. This is somewhat surprising since for SV, NSV, and NF, the errors $|(\mathbf{p}_n, \mathbf{q}_n) - (\mathbf{p}(t_n), \mathbf{q}(t_n))|$ grow like t^2. Also note that SF fails in exactly conserving energy, but its energy errors are much smaller than those associated with the remaining algorithms.

4. Integration of Kepler's problem by one-step methods. We now investigate theoretically some of the experimental findings presented above. Our analysis is not restricted to the RKN case and covers general one-step methods.

4.1. Some remarks on Kepler's problem. Let us begin by rewriting Kepler's problem in the compact form

$$\dot{\mathbf{Y}} = \mathbf{F}(\mathbf{Y}), \tag{4.1}$$

where $\mathbf{Y} = [p^1, p^2, q^1, q^2]^T$ and $\mathbf{F} = [\mathbf{f}^T, \mathbf{p}^T]^T$, with $\mathbf{f} = \mathbf{f}(\mathbf{q})$ the force (cf. (2.1)). The notation $\mathbf{G} = \mathbf{G}(\mathbf{Y})$ will be used to refer to the gradient ∇H of the Hamiltonian H with respect to $\mathbf{Y}$. Note that $\mathbf{F}$ and $\mathbf{G}$ are orthogonal at each point $\mathbf{Y}$ because H is an invariant quantity for (4.1).

We consider (4.1) in the region Ω of $\mathbf{Y}$-space covered by elliptic motions, i.e., the region where the energy H is less than 0 (so that escape to ∞ is not possible), and the angular momentum does not vanish (thus avoiding the case where the trajectory in $\mathbf{q}$-space degenerates into a straight segment). All solutions in Ω are periodic with a period

$$T = T(H) = 2\pi/\sqrt{(2|H|)^3}, \tag{4.2}$$

which only depends on the energy H. A reference for Kepler's problem is, e.g., [2, § 8E].

Let us, once and for all, fix an initial condition $\mathbf{Y}_0 \in \Omega$ and set $\mathbf{F}_0 = \mathbf{F}(\mathbf{Y}_0)$, $\mathbf{G}_0 = \mathbf{G}(\mathbf{Y}_0)$. We denote by Φ the one-period map φ_{T_0}, $T_0 = T(H(\mathbf{Y}_0))$. The analysis to follow relies heavily on the properties of the differential Φ_0' of Φ at $\mathbf{Y}_0$ (this is sometimes referred to as the monodromy operator of the periodic solution that goes through $\mathbf{Y}_0$).

LEMMA 1. *The differential Φ_0' is a rank-one modification of the identity given by*

$$\Phi_0' = I + \mathbf{W}_0 \mathbf{G}_0^T, \tag{4.3}$$

with $\mathbf{W}_0 = T'(H(\mathbf{Y}_0))\mathbf{F}_0$ a nonzero vector in $\mathbb{R}^4$ tangent at $\mathbf{Y}_0$ to the solution of Kepler's problem being investigated. Equivalently, Φ_0' is the linear operator in $\mathbb{R}^4$ such that, for vectors $\mathbf{V}$ orthogonal to $\mathbf{G}_0$,

$$\Phi_0' \mathbf{V} = \mathbf{V} \tag{4.4}$$

and

$$\Phi_0' \mathbf{G}_0 = \mathbf{G}_0 + (\mathbf{G}_0^T \mathbf{G}_0) \mathbf{W}_0. \tag{4.5}$$

Proof. We present two different proofs. The first is analytic and is due to R. D. Skeel. Set

$$\Phi(\mathbf{Y}) = \varphi_\tau(\varphi_{T(H(\mathbf{Y}))}(\mathbf{Y})) = \varphi_\tau(\mathbf{Y}),$$

where $\tau = T_0 - T(H(\mathbf{Y}))$ is a function of $\mathbf{Y}$. Then

$$\begin{aligned}\Phi'(\mathbf{Y}) = &= \varphi_\tau'(\mathbf{Y}) + \left(\frac{d}{d\tau}\varphi_\tau(\mathbf{Y})\right)(\nabla\tau)^T \\ &= \varphi_\tau'(\mathbf{Y}) - \left(\frac{d}{d\tau}\varphi_\tau(\mathbf{Y})\right) T'(H(\mathbf{Y}))\mathbf{G}(\mathbf{Y})^T,\end{aligned}$$

and, since φ_t as a function of t satisfies (4.1),

$$\Phi'(\mathbf{Y}) = \varphi'_\tau(\mathbf{Y}) - \mathbf{F}(\varphi_\tau(\mathbf{Y}))T'(H(\mathbf{Y}))\mathbf{G}(\mathbf{Y})^T.$$

Now evaluation at $\mathbf{Y}=\mathbf{Y}_0$ leads to $\tau=0$ and hence to φ_τ equal to the identity map so that $\varphi'_\tau(\mathbf{Y}_0)=I$ and (4.3) follows.

The second proof is more geometric. Consider a vector $\mathbf{V}$ orthogonal to $\mathbf{G}_0$ and consider the new initial condition $\tilde{\mathbf{Y}}_0=\mathbf{Y}_0+\varepsilon\mathbf{V}$ with ε small. Since the increment $\varepsilon\mathbf{V}$ is orthogonal to the energy gradient $\mathbf{G}_0$, the new energy $H(\tilde{\mathbf{Y}}_0)$ equals the old energy $H(\mathbf{Y}_0)$ and the new period $T(H(\tilde{\mathbf{Y}}_0))$ equals the old period T_0 (see (2.1)). Here and later "equal" is understood to mean "equal except for $O(\varepsilon^2)$ terms." Hence $\Phi(\tilde{\mathbf{Y}}_0)=\varphi_{T_0}(\tilde{\mathbf{Y}}_0)$ equals $\tilde{\mathbf{Y}}_0$, which implies, by the definition of differential, that $\Phi'_0(\varepsilon\mathbf{V})=\varepsilon V$. This proves (4.4). Assume now that the new initial condition is chosen to be $\tilde{\mathbf{Y}}_0=\mathbf{Y}_0+\varepsilon\mathbf{G}_0$. Now the new energy is in excess of $H(\mathbf{Y}_0)$ by an amount $\varepsilon(\mathbf{G}_0^T\mathbf{G}_0)$ and, accordingly, the new period is in excess from T_0 by an amount $\delta=\varepsilon T'(H(\mathbf{Y}_0))(\mathbf{G}_0^T\mathbf{G}_0)$. Hence after T_0 units of time $\mathbf{Y}$ has not had time to return to its initial position and rather lags *behind* by a vector $\delta\mathbf{F}_0$, because $\mathbf{F}_0$ is the velocity of the flow at $\mathbf{Y}_0$. This proves (4.5).

After (4.3) it is a simple matter to compute the Nth power of Φ'_0. This is given by

$$\Phi_0'^N = I + N\mathbf{W}_0\mathbf{G}_0^T. \tag{4.6}$$

This formula essentially says (cf. the second proof of the lemma above) that if $\tilde{\mathbf{Y}}_0$ is an initial condition of the form $\mathbf{Y}_0+\varepsilon\mathbf{V}$, ε small, then after N time increments of length T^0, the solution $\varphi_{NT^0}(\tilde{\mathbf{Y}}_0)$ that starts at $\tilde{\mathbf{Y}}_0$ differs from the solution $\varphi_{NT^0}(\mathbf{Y}_0)=\mathbf{Y}_0$ by terms $\varepsilon\mathbf{V}+\varepsilon N(\mathbf{G}_0^T\mathbf{V})\mathbf{W}_0$. The difference grows linearly with N; this growth is in the direction of $\mathbf{W}_0$, tangent at $\mathbf{Y}_0$ to the solution curve, and furthermore only depends on the initial deviation $\varepsilon\mathbf{V}$ through its component $\varepsilon(\mathbf{G}_0^T\mathbf{V})$ in the direction of $\mathbf{G}_0$.

4.2. Basic error estimate. Let us consider a smooth one-step method ψ_h for the numerical integration of Kepler's problem. This method is assumed to be convergent of order p, i.e., $\psi_h^n(\mathbf{Y}_0)-\varphi_h^n(\mathbf{Y}_0)=O(h^p)$ as $h\to 0$ with nh in a bounded time interval. Furthermore, we assume that the differentials (Jacobian matrices) $(\psi_h^n)'(\mathbf{Y})$ also converge with order p to the differential of the flow, i.e., $(\psi_h^n)'(\mathbf{Y})-(\varphi_h^n)'(\mathbf{Y})=O(h^p)$, $h\to 0$, nh bounded. This is automatically satisfied by most standard methods, including Runge-Kutta and Runge-Kutta-Nyström methods.

For simplicity, we only consider the case where the steplength h is of the form T_0/ν, with ν a positive integer, and look at the difference $\mathbf{E}_N$ between the numerical $\psi_h^{\nu N}(\mathbf{Y}_0)$ and theoretical $\varphi_h^{\nu N}(\mathbf{Y}_0)=\varphi_{NT_0}(\mathbf{Y}_0)=\mathbf{Y}_0$ after N periods of the motion. The extension to general values of h and times which are not whole multiples of T_0 is possible but messy, and provides no further insight.

Set $\Psi_h=\psi_h^\nu$ so that Ψ_h is the mapping that advances the numerical solution T_0 units of time. We can then write

$$\begin{aligned}\mathbf{E}_N &= \Psi_h^N(\mathbf{Y}_0)-\mathbf{Y}_0=\Psi_h(\Psi_h^{N-1}(\mathbf{Y}_0))-\Psi_h(\mathbf{Y}_0)+\mathbf{E}_1\\ &=\Psi'_h\mathbf{E}_{N-1}+O(\|\mathbf{E}_{N-1}\|^2)+\mathbf{E}_1\\ &=\Psi'_h\mathbf{E}_{N-1}+\mathbf{E}_1+O(h^{2p})\\ &=\Phi'_0\mathbf{E}_{N-1}+\mathbf{E}_1+O(h^{2p}).\end{aligned}$$

Here and later, the constant implied in the O symbol depends on N. By induction,

$$\mathbf{E}_N=[I+\Phi'_0+\cdots+\Phi_0'^{N-1}]\mathbf{E}_1+O(h^{2p}).$$

We apply (4.6) to conclude the following theorem.

THEOREM 1. *Under the hypothesis above,*

$$\mathbf{E}_N = N\mathbf{E}_1 + \tfrac{1}{2}(N^2 - N)(\mathbf{G}_0^T\mathbf{E}_1)\mathbf{W}_0 + O(h^{2p}). \tag{4.7}$$

In other words, except for $O(h^{2p})$ terms, the error $\mathbf{E}_N$ after N periods of the solution have been computed grows quadratically with N. The leading N^2 growth is in the direction tangent to the solution at $\mathbf{Y}_0$, corresponding to a phase error along the trajectory. After taking the inner product of (4.7) and the energy gradient $\mathbf{G}_0$, we conclude, in view of the orthogonality of $\mathbf{G}_0$ and $\mathbf{F}_0$, that the energy error after N periods is, except for $O(h^{2p})$ terms, N times the energy error after the first period (cf. Fig. 5).

4.3. The symplectic case. In this subsection we look at the situation where ψ_h is symplectic. The key fact for the analysis [14] is that, given an arbitrarily large positive integer q, it is possible to construct a modified autonomous Hamiltonian function $\tilde{H}_h = H + O(h^p)$ such that ψ_h is consistent of order q with the Hamiltonian problem with Hamiltonian $\tilde{H}_h$, i.e., $\psi_h - \varphi_{h,\tilde{H}_h} = O(h^{q+1})$, where $\varphi_{h,\tilde{H}_h}$ is the h-flow of the problem with Hamiltonian $\tilde{H}_h$. In other words, the mapping ψ_h we are using to integrate Kepler's problem can be seen (except for $O(h^{q+1})$ terms) as the exact flow of a nearby Hamiltonian problem with Hamiltonian $\tilde{H}_h$. Here we choose $q = 2p$. The computed points $\mathbf{Y}_n$, which are $O(h^p)$ away from the solution $\mathbf{Y}(t_n)$ of Kepler's problem, are only $O(h^{2p})$ away from the solution through $\mathbf{Y}_0$ of the modified problem. In particuular,

$$\Psi_h(\mathbf{Y}_0) - \varphi_{T_0,\tilde{H}_h}(\mathbf{Y}_0) = O(h^{2p})$$

and, by implication,

$$\tilde{H}_h(\Psi_h(\mathbf{Y}_0)) - \tilde{H}_h(\varphi_{T_0,\tilde{H}_h}(\mathbf{Y}_0)) = O(h^{2p}). \tag{4.8}$$

On the other hand, $\tilde{H}_h$ is a conserved quantity for the flow $\varphi_{T_0,\tilde{H}_h}$ so that (4.8) can be rewritten

$$\tilde{H}_h(\Psi_h(\mathbf{Y}_0)) - \tilde{H}_h(\mathbf{Y}_0) = O(h^{2p}). \tag{4.9}$$

Taylor expansion of the left-hand side of (4.9) yields

$$\tilde{H}_h(\Psi_h(\mathbf{Y}_0)) - \tilde{H}_h(\mathbf{Y}_0) = \mathbf{G}_0^{h^T}\mathbf{E}_1 + O(\|\mathbf{E}_1\|^2) = \mathbf{G}_0^{h^T}\mathbf{E}_1 + O(h^{2p}), \tag{4.10}$$

where $\mathbf{G}_0^h$ is the gradient of $\tilde{H}_h$ at $\mathbf{Y}_0$. Comparison of (4.9) with (4.10) shows that $\mathbf{G}_0^{h^T}\mathbf{E}_1 = O(h^{2p})$; the error after one period $\mathbf{E}_1$ is "almost" orthogonal to the gradient $\mathbf{G}_0^h$ of the modified energy $\tilde{H}_h$. Finally,

$$|\mathbf{G}_0^T\mathbf{E}_1| = |(\mathbf{G}_0 - \mathbf{G}_0^h)^T\mathbf{E}_1 + \mathbf{G}_0^{h^T}\mathbf{E}_1| \leqq \|\mathbf{G}_0 - \mathbf{G}_0^h\|\,\|\mathbf{E}_1\| + |\mathbf{G}_0^{h^T}\mathbf{E}_1| = O(h^{2p}).$$

Here we have used the fact that the derivatives of $\tilde{H}_h$ approximate the derivatives of H to the same order, $O(h^p)$, to which $\tilde{H}_h$ approximates H; see [14]. The last bound implies that for a symplectic method the component $\mathbf{G}_0^T\mathbf{E}_1$ of the error $\mathbf{E}_1$ is $O(h^{2p})$, so that in view of (4.7), we may state (cf. the dashed-dotted lines in Figs. 4 and 5) the following theorem.

THEOREM 2. *For a constant-stepsize symplectic method,*

$$\mathbf{E}_N = N\mathbf{E}_1 + O(h^{2p}).$$

Furthermore, the energy error satisfies

$$H(\Psi_h^N(\mathbf{Y}_0)) - H(\mathbf{Y}_0) = O(h^{2p}).$$

4.4. The nonsymplectic case. In this section we explain the experimental fact that the fourth-order method NSF employed in § 3 behaves as if its order were five. We consider a general one-step method as in § 4.3, but now assume that the order p is *even* ($p \geqq 2$) and, furthermore, that ψ_h possesses some symmetries. First, we assume that

$$(\mathbf{p}, \mathbf{q}) = \psi_h(\mathbf{p}_0, \mathbf{q}_0) \Rightarrow (-\mathbf{p}, \mathbf{q}) = \psi_{-h}(-\mathbf{p}_0, \mathbf{q}_0). \tag{4.11}$$

This symmetry holds for the flow of any problem of the form (2.1). Note that according to (2.2), RKN methods obey (4.11). Second, we assume that $(\mathbf{p}, \mathbf{q}) = \psi_h(\mathbf{p}_0, \mathbf{q}_0)$ inherits from φ_t the rotational symmetry of Kepler's problem. Again, this is true for standard methods.

Let us denote by $\mathbf{M}(t)$ the coefficient of the leading $O(h^p)$ term in the asymptotic expansion of the global error of the method ψ_h. It is well known that $\mathbf{M}$ satisfies the variational equation

$$\frac{d\mathbf{M}(t)}{dt} = J(t)\mathbf{M}(t) + \mathbf{L}(t), \tag{4.12}$$

where $J(t)$ is the Jacobian of the vector field evaluated at the theoretical solution $\varphi_t(\mathbf{Y}_0)$, and $\mathbf{L}(t)$ is the coefficient of the leading $O(h^{p+1})$ term in the expansion of the local error. We need the following lemma.

LEMMA 2. *Let $dy/dt = f(y)$ be any smooth differential system with a conserved quantity H. Let*

$$\frac{dm(t)}{dt} = J(t)m(t) + l(t) \tag{4.13}$$

be the corresponding variational equation at a solution of the system. Then,

$$\frac{d}{dt}((\nabla H)^T m) = (\nabla H)^T l.$$

Proof. From (4.13),

$$\frac{d}{dt}(\nabla H)^T m = (\nabla H)^T J(t) m + (\nabla H)^T l + \left(\frac{d}{dt}\nabla H\right)^T m.$$

Differentiation with respect to y of the identity $(\nabla H)^T f \equiv 0$ and evaluation of the result at the solution of $dy/dt = f(y)$ leads to $(\nabla H)^T J + (d\nabla H/dt)^T = 0$, and the proof is complete.

The application of Lemma 2 to (4.12) reveals that

$$\frac{d}{dt}\mathbf{G}^T\mathbf{M} = \mathbf{G}^T\mathbf{L}.$$

We integrate over one period to get

$$\mathbf{G}_0^T\mathbf{M}(T_0) = \int_0^{T_0} \mathbf{G}^T\mathbf{L}\, dt. \tag{4.14}$$

Now let us express $\mathbf{G}^T\mathbf{L}$ in terms of polar coordinates r, $\dot{r}$, θ, $\dot{\theta}$. The rotational symmetry assumed above implies that $\mathbf{G}^T\mathbf{L}$ does not depend on θ. By conservation of angular momentum along the solution, $\dot{\theta}$ can be expressed in terms of r. Thus, in (4.14) the integrand is a function of r and $\dot{r}$. Furthermore, $\mathbf{G}^T\mathbf{L}$ must be *odd* in $\dot{r}$. This is because $h^{p+1}\mathbf{G}^T\mathbf{L}$ is the leading term in energy error after one step, and by (4.11) such an error remains invariant when h is changed into $-h$ and $\dot{r}$ into $-\dot{r}$, while keeping r constant. Now, as t increases from 0 to T_0, the solution takes each value of r twice with opposite values of $\dot{r}$. (This occurs when the moving point passes through points in Kepler's ellipse that are symmetric with respect to the major axis.) Hence, the integral in (4.14) vanishes and $\mathbf{G}_0^T\mathbf{M}(T_0)=0$, i.e., $\mathbf{G}_0^T\mathbf{E}_1=O(h^{p+1})$. When this information is taken into (4.7) we see that $\mathbf{E}_N$ contains $O(Nh^p)$ and $O(N^2h^{p+1})$ terms. Assume that N is very large. The terms Nh^p, N^2h^{p+1} are of the same size when $h=1/N$, which is unrealistically small since for $h=1/N$, $Nh^p=N^2h^{p+1}=N^{-p+1}\ll 1$. Hence, for realistic choices of h, N^2h^{p+1} is much larger than Nh^p, and the method behaves as an order $p+1$ method with a large N^2 error constant.

4.5. Variable steps. Let us now study the situation for the variable-step integrators. In the experiments we only used *one* initial condition $\mathbf{Y}_0$. The choice of initial condition and tolerance determines the sequence of stepsizes $h_1, h_2, \ldots$ used in the integration. In a "thought experiment," let us imagine that even if other neighbouring initial conditions had been used, we would have still employed the same sequence $h_1, h_2, \ldots$ used for $\mathbf{Y}_0$, rather than letting the step-changing mechanism dictate the choices of stepsizes. This is actually a recommended procedure that ensures that the output of an automatic code is a smooth function of the initial data [8, § II.5]. In our context it also ensures that, if a symplectic formula ψ_h is used, then the transformation $\psi_{h_m}\cdots\psi_{h_1}$, which advances the solution from time $t=0$ to time $t_m=h_1+\cdots+h_m$, is indeed a symplectic transformation. In extending the analysis above to the variable-step experiments, we encounter some difficulties. Previously, we used the fact that we advanced the solution from time $t=0$ to time $t=NT_0$ by iterating N times an operator Ψ_h that advances the solution T_0 units of time. This is not quite true now; it is possible for T_0 not to be a steppoint t_m. But even if it is a steppoint, the sequence of stepsizes employed to go around the orbit in the second, third, ... period is likely to be slightly different from the sequence used in the first period. These difficulties will be ignored for the analysis: we assume that T_0 is a steppoint t_m, and that the sequence of stepsizes used to cover the nth period $(n-1)T_0\leqq t\leqq nT_0$ is just a duplicate of the sequence used to cover the first period $0\leqq t\leqq T_0$. These assumptions are "almost" satisfied for small tolerances; see [20], where it is rigorously shown that, essentially, variable-step algorithms employ a steplength that only depends on the current point in phase space so that, for periodic problems, stepsizes repeat themselves periodically. With our assumptions, the solution after N orbits is given by $\Psi_h^N(\mathbf{Y}_0)$, where $\Psi_h=\psi_{h_m}\cdots\psi_{h_1}$. Then the analysis in § 4.2 leading to Theorem 1 holds with h, the maximum stepsize. Furthermore, the cancellation described in § 4.4 also holds. The functions $\mathbf{M}$ and $\mathbf{L}$ still make sense [16] provided that the stepsizes satisfy

$$h_n=\gamma(t_n)h+O(h^2),$$

with γ a stepsize function; (4.12) must be replaced by [16]

$$\frac{d\mathbf{M}(t)}{dt}=J(t)\mathbf{M}(t)+\gamma(t)\mathbf{L}(t),$$

and (4.14) becomes

$$\mathbf{G}_0^T\mathbf{M}(T_0)=\int_0^{T_0}\gamma^p\mathbf{G}^T\mathbf{L}\,dt.$$

From here we conclude that $\mathbf{G}_0^T\mathbf{E}_1=O(h^{p+1})$ under the extra hypothesis that γ takes the same value as the moving body passes through points in configuration space that are mutually symmetric with respect to the major axis. This hypothesis is certainly reasonable.

On the other hand, the material in § 4.3 does not appear to be extensible to the variable-step situation. Indeed, the experiments indicate that it *cannot* be extended. When trying to extend the analysis in § 4.3 to variable steps, we encounter the difficulty that the modified Hamiltonian $\tilde{H}_h$ depends on the steplength. The computed $(\mathbf{p}_1, \mathbf{q}_1)$ is close to the solution S_1 of the system with Hamiltonian $\tilde{H}_{h_1}$ which at $t=0$ passes through $(\mathbf{p}_0, \mathbf{q}_0)$. The computed $(\mathbf{p}_2, \mathbf{q}_2)$ is close to the solution S_2 of the system associated with $\tilde{H}_{h_2}$ that at $t=h_1$ passes through $(\mathbf{p}_1, \mathbf{q}_1)$, etc. Clearly, $S_1 \neq S_2$ (unless $\tilde{H}_{h_1} = \tilde{H}_{h_2}$) and we do not have a single trajectory near which the computed points stay. There is not a single pseudoenergy $\tilde{H}_h$ "almost" conserved by the numerical points, and nothing can be said of the projection $\mathbf{G}_0^T\mathbf{E}_1$, whose smallness is the key to the success of the symplectic constant-stepsize integrators.

4.6. Remarks and extensions. The fact that for Kepler's problem standard integrators lead to quadratic error growth while symplectic constant-stepsize integrators lead to linear error growth has been noted before in the literature; see Kinoshita, Yoshida, and Nakay [10]. In [22], Yoshida studies the energy error in the symplectic integration of Kepler's problem. His analysis is only formal and, like ours, resorts to a modified Hamiltonian $\tilde{H}_h$. Yoshida assumes that the modified Hamiltonian can be chosen to satisfy

$$\psi_h - \phi_{h,\tilde{H}_h} = 0, \tag{4.15}$$

i.e., that a modified Hamiltonian problem exists so that the computed points exactly solve the modified problem. However, it is known that for nonlinear problems, while it is possible to construct a divergent formal power series for $\tilde{H}_h$ fulfilling (4.15), no actual function $\tilde{H}_h$ can satisfy (4.15). Therefore, the analysis in [22] is only of heuristic value. (Note that rather than (4.15), we assumed only that $\psi_h - \phi_{h,\tilde{H}_h} = O(h^{2p})$.)

On the other hand, the ideas used in the analysis in this section are not restricted to Kepler's problem. For instance, in §§ 4.2–4.4, it is enough to assume that all solutions of the problem being integrated are periodic with a period that only depends on the value of the energy H (and actually changes with H). These assumptions are satisfied by all nonlinear one-degree-of-freedom oscillators, such as the well-known pendulum equation. Therefore, the conclusions in §§ 4.2–4.4 hold for such oscillators. This proves Conjecture 3 in § IV.6 D of Stoffer's thesis [18], which states that for nonlinear oscillators, standard methods have quadratic error growth, and that symplectic methods produce errors that only grow linearly. (Nonlinearity is essential to guarantee a nontrivial dependence of the period on the energy, leading to $\mathbf{W}_0 \neq \mathbf{0}$ in (4.6).)

5. Conclusions. Let us summarize our findings.

(i) The experiments with Kepler's problem reported above and experiments with other Hamiltonian problems (not reported in this paper) reveal that constant-stepsize symplectic integrators can be more efficient than variable-step codes. This provides motivation for the further study of symplectic integration. Comparisons between symplectic and nonsymplectic formulae presented so far in the literature (see [14] for references) have concentrated on constant stepsizes. Our experiments indicate that it is reasonable to expect that, in the future, symplectic software can be developed which outperforms, on Hamiltonian problems, standard variable-step codes. The paper by Herbst and Ablowitz [9], written after the present work was completed, provides a dramatic example of a simple symplectic algorithm outperforming NAG library software.

(ii) The advantages of using symplectic formulae are lost when these formulae are used in a variable-stepsize environment. This came as a surprise to us. However, after completing this work, we discovered that in 1988, Stoffer [19] had argued that symplectic integrators should not be used with variable stepsizes. His argument is as follows. Integrating a system of ODEs $d\mathbf{y}/dt = \mathbf{f}(\mathbf{y})$ with a variable-step algorithm is "equivalent" [20] to integrating, with constant stepsizes, a transformed problem $d\mathbf{y}/d\tau = r(\mathbf{y})\mathbf{f}(\mathbf{y})$, where the new time τ is related to the old time by $dt/d\tau = r(\mathbf{y})$. The transformed system is not Hamiltonian, even if the original system is, so that the advantages of symplecticness are lost in the transformation. An alternative argument to justify the failure of variable-step symplectic algorithms has been put forward in [14]. A key property of symplectic formulae ψ_h for a Hamiltonian problem with Hamiltonian H is the existence of a modified Hamiltonian $\tilde{H}_h$ in such a way that ψ_h "almost" coincides with the h-flow $\phi_{h,\tilde{H}_h}$ of $\tilde{H}_h$. "Almost" means that, given any large integer q, $\tilde{H}_h$ can be found in such a way that $\psi_h - \phi_{h,\tilde{H}_h} = O(h^{q+1})$. With constant stepsizes, it is possible to interpret the error in a "backward" way: a numerically calculated solution corresponding to H is "almost" an exact solution of a neighbouring Hamiltonian $\tilde{H}_h$. In § 4.5 we saw how such a backward-error analysis interpretation fails in a variable-stepsize situation. An additional reference useful in connection with variable steps for symplectic integrators is [17].

(iii) For the particular cases of Kepler's problem and nonlinear one-degree-of-freedom oscillators, a complete analysis has been presented of the performance of symplectic and nonsymplectic integrators. It has been shown that the advantages of symplectic integrators include not only better qualitative behaviour, but also better quantitative properties in the error growth mechanism.

REFERENCES

[1] L. Abia and J. M. Sanz-Serna, *Partitioned Runge-Kutta methods for separable Hamiltonian problems*, Math. Comput., 1993, to appear.

[2] V. I. Arnold, *Mathematical Methods of Classical Mechanics*, 2nd ed., Springer-Verlag, New York, 1989.

[3] M. P. Calvo and J. M. Sanz-Serna, *Order conditions for canonical Runge-Kutta-Nyström methods*, BIT, 3 (1992), pp. 131-142.

[4] P. J. Channell and C. Scovel, *Symplectic integration of Hamiltonian systems*, Nonlinearity, 3 (1990), pp. 231-259.

[5] J. R. Dormand, M. E. A. El-Mikkawy, and P. J. Prince, *Families of Runge-Kutta-Nyström formulae*, IMA J. Numer. Anal., 7 (1987), pp. 235-250.

[6] J. R. Dormand and P. J. Prince, *A family of embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 6 (1980), pp. 19-268.

[7] K. Feng, *Difference schemes for Hamiltonian formalism and symplectic geometry*, J. Comput. Math., 4 (1986), pp. 279-289.

[8] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations* I, *Nonstiff Problems*, Springer-Verlag, Berlin, 1987.

[9] B. M. Herbst and M. J. Ablowitz, *Numerical homoclinic instabilities in the sine-Gordon equation*, Quaestiones Mathematicae, 15 (1992), pp. 345-363.

[10] H. Kinoshita, H. Yoshida, and H. Nakay, *Symplectic integrators and their application to dynamical astronomy*, Celestial Mech., 50 (1991), pp. 59-71.

[11] D. Okunbor and R. D. Skeel, *An explicit Runge-Kutta-Nyström method is canonical if and only if its adjoint is explicit*, SIAM J. Numer. Anal., 19 (1992), pp. 521-527.

[12] R. Ruth, *A canonical integration technique*, IEEE Trans. Nuclear Sci., 30 (1983), pp. 2669-2671.

[13] J. M. Sanz-Serna, *Runge-Kutta schemes for Hamiltonian systems*, BIT, 28 (1988), pp. 877-883.

[14] ———, *Symplectic Integrators for Hamiltonian problems: An overview*, Acta Numerica, 1 (1992), pp. 243-286.

[15] J. M. Sanz-Serna and L. Abia, *Order conditions for canonical Runge-Kutta schemes*, SIAM J. Numer. Anal., 28 (1991), pp. 1081-1096.

[16] L. F. Shampine, *The step sizes used by one-step codes for* ODE's, Appl. Numer. Math., 1 (1985), pp. 95-106.

[17] R. D. Skeel and C. W. Gear, *Does variable step size ruin a symplectic integrator?*, Phys. D, 60 (1992), pp. 311-313.

[18] D. M. STOFFER, *Some geometric and numerical methods for perturbed integrable systems*, Ph.D. thesis, Eidgenoessische Technische Hochschule (ETH), Zürich, 1988.

[19] ———, *On reversible and canonical integration methods*, Res. Rep. No. 88-05, Applied Mathematics, Eidgenoessische Technische Hochschule (ETH) Zürich, 1988.

[20] D. M. STOFFER AND K. NIPP, *Invariant curves for variable step size integrators*, BIT, 31 (1991), pp. 169–180.

[21] Y. B. SURIS, *Canonical transformations generated by methods of Runge–Kutta type for the numerical integration of the system $x'' = -\partial U/\partial x$*, Zh. Vychisl, Mat. i Mat. Fiz., 29 (1989), pp. 202-211. (In Russian.)

[22] H. YOSHIDA, *Conserved quantities of symplectic integrators for Hamiltonian systems*, preprint.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 953-970, July 1993

ITERATIVE DEFECT CORRECTION AND MULTIGRID ACCELERATED EXPLICIT TIME STEPPING SCHEMES FOR THE STEADY EULER EQUATIONS*

MARIE-HÉLÈNE LALLEMAND† AND BARRY KOREN‡

Abstract. Analytical and experimental convergence results are presented for a novel pseudo-unsteady solution method for higher-order accurate upwind discretizations of the steady Euler equations. Comparisons are made with an existing pseudo-unsteady solution method. Both methods make use of nonlinear multigrid for acceleration and nested iteration for the fine-grid initialization. The new method uses iterative defect correction. Analysis shows that it not only has better stability but it also has better smoothing properties. The analytical results are confirmed by numerical experiments, which show better convergence and efficiency.

Key words. defect correction, multigrid, explicit time stepping, Euler equations

AMS subject classifications. 65B05, 65N10, 65N20, 65N30, 65N40, 76G15

1. Introduction.

1.1. Equations. The equations considered are the steady, two-dimensional, compressible Euler equations

$$\frac{\partial F(W)}{\partial x}+\frac{\partial G(W)}{\partial y}=0, \tag{1.1}$$

where

$$W=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho e\end{pmatrix},\quad F(W)=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho u(e+p/\rho)\end{pmatrix},\quad G(W)=\begin{pmatrix}\rho v\\ \rho vu\\ \rho v^2+p\\ \rho v(e+p/\rho)\end{pmatrix}. \tag{1.2}$$

Assuming a perfect gas, the total energy e satisfies $e=p/(\rho(\gamma-1))+\frac{1}{2}(u^2+v^2)$. The ratio of specific heats γ is assumed to be constant.

1.2. Spatial discretization. The computational grid is obtained by a hybrid finite element—finite volume partition. A (possibly unstructured) finite element triangularization is used as the basic partition. A cell-centered finite volume partition is derived from the finite element partition by connecting the centers of the triangle sides in the manner illustrated in Fig. 1. The finite volume grid gives us the easy possibility of grouping together the nodes associated with contiguous finite volumes. If we take unions of control volumes this results in a new coarser mesh. Repetition of this operation gives coarser and coarser meshes. For details about this hybrid way of constructing finite volume grids, see [1]. For applications in single-grid Euler and Navier-Stokes flow computations, we refer to [5] and [23], respectively. For details about the coarsening process (multilevel gridding), we refer to [16].

* Received by the editors April 17, 1989; accepted for publication (in revised form) August 4, 1992. This work was performed as a cooperation between l'Institut National de Recherche en Informatique et en Automatique (INRIA) and the Centrum voor Wiskunde en Informatica (CWI). The investigations were supported by the Centre National d'Etudes Spatiales (CNES), and by the European Space Agency (ESA) through Avions Marcel Dassault-Bréguet Aviation (AMD-BA).

† L'Institut National de Recherche en Informatique et en Automatique, Domaine de Voluceau, Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France.

‡ Centrum voor Wiskunde en Informatica, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands.

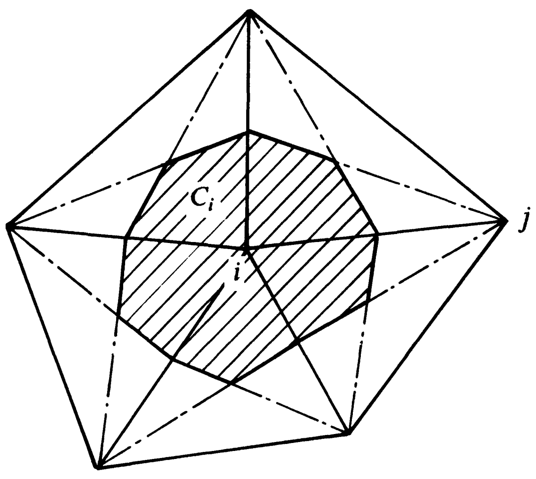


FIG. 1. *Finite volume C_i.*

On the finest grid, for all finite volumes C_i, $i=1,2,\dots,N$, we consider the integral form

$$(1.3)\qquad \oint_{\partial C_i} (F(W)n_x + G(W)n_y)\,ds = 0, \qquad i=1,2,\dots,N,$$

with n_x and n_y the x- and y-component of the outward unit normal on the volume boundary ∂C_i. For the Euler equations, because of their rotational invariance, (1.3) may be rewritten as

$$(1.4)\qquad \oint_{\partial C_i} T^{-1}(n_x, n_y)F(T(n_x, n_y)W)\,ds = 0, \qquad i=1,2,\dots,N,$$

where $T(n_x, n_y)$ is the rotation matrix

$$(1.5)\qquad T(n_x, n_y) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & n_x & n_y & 0 \\ 0 & -n_y & n_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

For simplicity, we assume the flux to be constant across each bisegment ∂C_{ij} of the boundary ∂C_i, where $\partial C_{ij} = \partial C_i \cap \partial C_j$ is the common boundary between the neighboring volumes C_i and C_j (Fig. 2(a)). Hence, $\partial C_i = \bigcup \partial C_{ij}, j=1,2,\dots,n_i$, with n_i the number of neighboring volumes C_j. (In the example of Fig. 1; $n_i=5$.) Since we have assumed that the flux is constant along ∂C_{ij}, it is equal to the flux across the straight segment

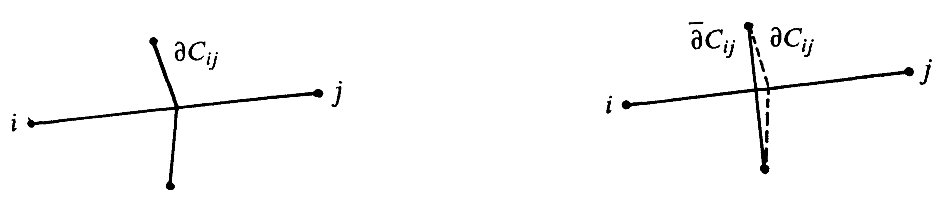


FIG. 2. *Segments in between finite volumes C_i and C_j.* (a) *Bisegment ∂C_{ij}.* (b) *Straight segment $\bar{\partial} C_{ij}$.*

$\bar{\partial} C_{ij}$ connecting the two extreme points of ∂C_{ij} (Fig. 2(b)). If we introduce the outward unit normal $\bar{n}_{ij} = ((\bar{n}_x)_{ij}, (\bar{n}_y)_{ij})^T$ along each $\bar{\partial} C_{ij}, j = 1, 2, \ldots, n_i$, with the assumption of a constant flux, the contour integral (1.4) can be rewritten as the sum

$$\sum_{j=1}^{n_i} \bar{T}_{ij}^{-1} F(\bar{T}_{ij} W_{ij}) l_{ij} = 0, \qquad i = 1, 2, \ldots, N, \tag{1.6}$$

where $\bar{T}_{ij} = T((\bar{n}_x)_{ij}, (\bar{n}_y)_{ij})$, where W_{ij} is some value of W depending on for instance W_i and W_j, and where l_{ij} is the length of the segment $\bar{\partial} C_{ij}$.

Crucial in (1.6) is the way in which the cell face flux $F(\bar{T}_{ij} W_{ij})$ is evaluated. For this we use an upwind scheme that follows the Godunov principle [8], which assumes that the constant flux vector along each segment $\bar{\partial} C_{ij}$ is determined only by a uniformly constant left and right cell face state (W_{ij}^l and W_{ij}^r). The one-dimensional Riemann problem, which then arises at each cell face, is solved in an approximate way. With this, (1.6) can be further written as

$$\sum_{j=1}^{n_i} \bar{T}_{ij}^{-1} \Phi(\bar{T}_{ij} W_{ij}^l, \bar{T}_{ij} W_{ij}^r) l_{ij} = 0, \qquad i = 1, 2, \ldots, N, \tag{1.7}$$

where Φ denotes the approximate Riemann solver. Several approximate Riemann solvers exist (see, for example, [20] and [22]). In this paper, without any particular motivation, we restrict ourselves to the application of Osher's approximate Riemann solver [20].

The flux evaluation, and so the space discretization, may be either first- or higher-order accurate. First-order accuracy is obtained in the standard way; at each finite volume wall, the left and right cell face state, which must be inserted in the numerical flux function, are taken equal to those in the corresponding adjacent volumes

$$W_{ij}^l = W_i, \tag{1.8a}$$

$$W_{ij}^r = W_j. \tag{1.8b}$$

Whereas the first-order accurate discretization is applied at all levels, the higher-order discretization is applied at the finest grid only, using the finite element partition existing there. Higher-order accuracy is obtained with a MUSCL approach [18]. Here, W_{ij}^l and W_{ij}^r are derived from linear interpolations. On each volume C_i around the triangle-vertex i an approximate gradient, denoted by $(\bar{\nabla} W)_i$, is derived by integrating the gradient of the linear interpolant of W over all the triangles that have i as a vertex:

$$(\bar{\nabla} W)_i = \left[\left(\frac{\bar{\partial} W}{\partial x} \right)_i, \left(\frac{\bar{\partial} W}{\partial y} \right)_i \right]^T, \tag{1.9a}$$

with

$$\left(\frac{\bar{\partial} W}{\partial x} \right)_i = \frac{\int_{\text{supp}\,(i)} (\partial W / \partial x)\, dx\, dy}{\int_{\text{supp}\,(i)} dx\, dy}, \tag{1.9b}$$

$$\left(\frac{\bar{\partial} W}{\partial y} \right)_i = \frac{\int_{\text{supp}\,(i)} (\partial W / \partial y)\, dx\, dy}{\int_{\text{supp}\,(i)} dx\, dy}. \tag{1.9c}$$

Here, supp (i) denotes the union of triangles which have i as a vertex. Then for each pair of neighboring vertices (i, j) we compute the extrapolated values

$$W_{ij}^l = W_i + \tfrac{1}{2} (\bar{\nabla} W)_i \cdot \overline{ij}, \tag{1.10a}$$

$$W_{ij}^r = W_j - \tfrac{1}{2} (\bar{\nabla} W)_j \cdot \overline{ij}. \tag{1.10b}$$

On equidistant grids, this higher-order accurate discretization can be formally proved to be second-order accurate. The proof is still valid for nearly equidistant grids. In this paper we do not analyze orders of accuracy; the discretization is already known. It has been described in more detail in other papers; see, for example, [7].

In order to ensure monotonicity, while preserving the higher-order accuracy in smooth flow regions, the higher-order values W^l_{ij} and W^r_{ij} according to (1.10) can be replaced by limited values that do not affect the order of accuracy.

1.3. Existing solution method. To solve the steady discretized system (1.7), we consider the unsteady, semidiscrete system of ordinary differential equations

$$\frac{dW_i}{dt} = R_i, \qquad i = 1, 2, \ldots, N. \tag{1.11}$$

The natural choice for R_i is

$$R_i = \frac{-1}{A_i} \sum_{j=1}^{n_i} \bar{T}_{ij}^{-1} \Phi(\bar{T}_{ij} W^l_{ij}, \bar{T}_{ij} W^r_{ij}) l_{ij}, \tag{1.12}$$

where A_i is the area of finite volume C_i.

As an upwind analogue to Jameson's central method [13], in [16] and [17] an explicit four-stage Runge–Kutta (RK4) scheme is applied for the temporal integration of (1.11)-(1.12). The benefits of the upwind analogue are evident: better shock capturing, greater robustness, and no tuning of explicitly added artificial viscosity. Similarly, just as in [13], in [16], multigrid is applied for accelerating the solution process. Furthermore, just as in [13], time accuracy is not pursued, and optimal Runge–Kutta coefficients are applied to get good stability as well as good smoothing properties. It seems that the solution method presented in [16] is already competitive with Jameson's method, without the introduction of a further acceleration technique such as, for example, residual averaging.

It is interesting that the upwind analogue allows a further efficiency improvement by exploitation of the direct availability of the corresponding first-order upwind discretization, with its better stability and smoothing properties. Since a first-order central discretization is not readily available, a standard central method does not easily allow this improvement.

2. Novel solution method.

2.1. Explicit time stepping. Compared with the existing solution method, the new solution method only uses a more extensive right-hand side in the explicit time stepping scheme. The extension consists of two first-order upwind defects; one which is evaluated at each stage of the multistage scheme, and another which is kept frozen during a fixed number of ν_t RK4 time steps ($\nu_t \geqq 1$), and which compensates for the other first-order defect by its opposite sign. Furthermore, significantly, the higher-order defect is kept frozen as well during ν_t RK4 steps. The four-stage time stepping scheme is written as

```
(2.1a)    $W_i^{0,4} := W_i^{0,0}, \quad i = 1, 2, \ldots, N$
          for $\nu$ from 1 to $\nu_t$ do
(2.1b)      $W_i^{\nu,0} := W_i^{\nu-1,4}, \quad i = 1, 2, \ldots, N$
            for $k$ from 1 to 4 do
(2.1c)        $W_i^{\nu,k} := W_i^{\nu,0} + \Delta t_i \alpha_k R_i^{\nu,k-1}, \quad i = 1, 2, \ldots, N$
            enddo
          enddo
```

Here, ν is the time step number, k the stage number, Δt_i the local time step, and α_k the kth Runge-Kutta coefficient. In the existing higher-order method the right-hand side $R_i^{\nu,k-1}$ is

$$(2.2)\qquad R_i^{\nu,k-1} = \frac{-1}{A_i}\sum_{j=1}^{n_i} \bar{T}_{ij}^{-1}\Phi(\bar{T}_{ij}(W_{ij}^l)^{\nu,k-1}, \bar{T}_{ij}(W_{ij}^r)^{\nu,k-1})l_{ij},$$

with $(W_{ij}^l)^{\nu,k-1}$ and $(W_{ij}^r)^{\nu,k-1}$ higher-order accurate. So, nothing is kept frozen in the existing method's right-hand side. For the novel method, we take

$$(2.3)\qquad \begin{aligned} R_i^{\nu,k-1} = \frac{-1}{A_i}\sum_{j=1}^{n_i} \bar{T}_{ij}^{-1}(&\Phi(\bar{T}_{ij}W_i^{\nu,k-1}, \bar{T}_{ij}W_j^{\nu,k-1}) - \Phi(\bar{T}_{ij}W_i^{0,0}, \bar{T}_{ij}W_j^{0,0}) \\ &+ \Phi(\bar{T}_{ij}(W_{ij}^l)^{0,0}, \bar{T}_{ij}(W_{ij}^r)^{0,0}))l_{ij}, \end{aligned}$$

where only $(W_{ij}^l)^{0,0}$ and $(W_{ij}^r)^{0,0}$ are higher-order accurate. The frozen first-order cell face states ($W_i^{0,0}$ and $W_j^{0,0}$) and the frozen higher-order cell face states ($(W_{ij}^l)^{0,0}$ and $(W_{ij}^r)^{0,0}$) are updated in an additional outer iteration, which will be explained in the next section. In the following, for convenience, $W^{\nu,4}$ will also be denoted as $W_{\mathrm{RK4}}(\nu, R^{0,0}, W^{0,0})$.

2.2. Complete solution method. The novel solution method is of defect correction type [2]. Though defect correction iteration is not as necessary for a pseudo-unsteady solution method as it is for a solution method that directly tackles steady discretized equations [12], [14], [15], it may lead to an improved efficiency. In § 3, we will show that the defect correction method proposed here can take advantage of a *greater stability* domain (larger local time steps) guaranteed by the first-order defects in the right-hand side. Furthermore, we will show that with multigrid as an acceleration technique, advantage can also be taken of *better smoothing* properties.

The new solution method can be divided into two successive stages. The first stage is *nested iteration* [10, p. 98], also called *full multigrid* (FMG) method [4], which is applied to obtain a good initial solution on the finest grid. The second stage is an *iterative defect correction* (IDeC) method [2], [10, p. 282], which is used to iterate until the higher-order accurate solution is obtained. The initial solution for the defect correction process is the solution obtained by the nested iteration. The inner iteration of both stages is a *nonlinear multigrid* method [10, p. 181], viz. the *full approximation storage* (FAS) algorithm [3], [4]. In the following sections we discuss successively: the nested iteration (§ 2.2.1), the iterative defect correction method (§ 2.2.2), and the building block of these two iterations: the nonlinear multigrid iteration (§ 2.2.3).

2.2.1. Nested iteration. To apply multigrid we construct a nested set of grids. Let $\Omega_1, \Omega_2, \dots, \Omega_L$ be a sequence of nested grids with Ω_1 the coarsest and Ω_L the finest grid. (For a description of the coarsening rule applied here, we refer to [16].) The nested iteration (FMG) starts with a user-defined initial estimate of W_1: the solution on the coarsest grid Ω_1. To obtain an initial solution on Ω_2, the solution on Ω_1 is first improved by a few FAS cycles. (The number of FAS cycles that is applied in each FMG step can be either fixed, $\nu_{\mathrm{FAS}} = \text{constant}$, or dependent on the residual.) After this, the improved solution W_1 is prolongated to Ω_2. The process is repeated until Ω_L has been reached.

The prolongation of the solution can be the simple piecewise constant prolongation I_{l-1}^l, $2 \leqq l \leqq L$, or it can be a smoother one. If we denote the area of finite volume C_i^l at level l by A_i^l, and the number of neighboring volumes C_j^l of C_i^l by n_i^l, a smooth prolongation operator $\mathcal{I}^l_{l-1}$ is defined by

$$(2.4)\qquad (\mathcal{I}^l_{l-1}(W_{l-1}))_i \equiv \frac{A^l_i(I^l_{l-1}(W_{l-1}))_i + \sum_{j=1}^{n^l_i} A^l_j(I^l_{l-1}(W_{l-1}))_j}{A^l_i + \sum_{j=1}^{n^l_i} A^l_j}, \qquad 2 \leqq l \leqq L.$$

We note that since I^l_{l-1} strictly obeys the physical conservation laws by the prolongation of cell-integrated amounts of mass, momentum, and energy, $\mathcal{I}^l_{l-1}$ is strictly conservative as well.

2.2.2. Iterative defect correction. Let $\mathcal{F}^1_L(W_L) = 0$ and $\mathcal{F}^+_L(W_L) = 0$ denote the first-order and higher-order discretized Euler equations, respectively, on the finest grid. Then, IDeC can be written as

$$(2.5a)\qquad \mathcal{F}^1_L(W^0_L) = 0,$$

$$(2.5b)\qquad \mathcal{F}^1_L(W^n_L) = \mathcal{F}^1_L(W^{n-1}_L) - \mathcal{F}^+_L(W^{n-1}_L), \qquad n = 1, 2, \dots, n_{\mathrm{IDeC}},$$

where W^0_L is the solution yielded by FMG. From (2.5b), it is immediately clear that at convergence ($W^n_L = W^{n-1}_L = W_L$), we have solved the higher-order discretized Euler equations $\mathcal{F}^+_L(W_L) = 0$. Therefore, we emphasize that the present defect correction method is *not* mixed defect correction iteration [11]. (A mixed defect correction method would yield a solution whose accuracy is not well defined; its solution would be a vague mixture of the first-order and higher-order accurate solutions.) Though both theory [6] and practice [6] show that IDeC gives poor convergence of the residual, theory [9], [10, p. 282] and practice [12], [15] also show that for smooth problems, a single IDeC cycle ($n_{\mathrm{IDeC}} = 1$) is sufficient to obtain second-order solution accuracy. Furthermore, for solutions with discontinuities, a few IDeC cycles ($n_{\mathrm{IDec}} \approx 5$) may improve the accuracy to a sufficient extent [12], [14]. In summary, for both smooth and nonsmooth flow problems, numerical experiments with IDeC show this phenomenon of *slow convergence* but of *fast solution improvement* [6], [12], [14], [15]; a phenomenon that is understood by theory [6], [9], [10, p. 282].

In each IDeC cycle we must solve a first-order system with an appropriate right-hand side. From [14] it is known that it is inefficient to solve this system very accurately. With a steady approach, application of only a single FAS cycle per IDeC cycle appears to be the most efficient strategy in [14]. In this paper, with our unsteady approach, we will re-investigate what is the most efficient number of FAS cycles per IDeC cycle (see § 3).

2.2.3. Nonlinear multigrid iteration. Let us denote by $(W_l)_{V(\nu_{\mathrm{pre}},\nu_{\mathrm{post}})}(\nu_{\mathrm{FAS}}, R_l, W^0_l)$ the solution obtained on level l, after ν_{FAS} FAS $V(\nu_{\mathrm{pre}}, \nu_{\mathrm{post}})$-cycles have been applied to $\mathcal{F}^1_l(W_l) = R_l$, with initial solution W^0_l. A single FAS $V(\nu_{\mathrm{pre}}, \nu_{\mathrm{post}})$-cycle on level l, $1 \leqq l \leqq L$, is then recursively defined by the following successive steps:

(i) Improve on the grid Ω_l the initial solution W^0_l by applying ν_{pre} RK4 steps to

$$(2.6)\qquad \mathcal{F}^1_l(W_l) = R_l.$$

Let us denote the resulting solution $(W_l)_{\mathrm{RK4}}(\nu_{\mathrm{pre}}, R_l, W^0_l)$ by $\bar{W}_l$.

(ii) Coarse-grid correction step: Approximate on the underlying coarser grid Ω_{l-1} the solution of

$$(2.7a)\qquad \mathcal{F}^1_{l-1}(W_{l-1}) = R_{l-1},$$

$$(2.7b)\qquad R_{l-1} = \mathcal{F}^1_{l-1}(I^{l-1}_l(\bar{W}_l)) - \tilde{I}^{l-1}_l(\mathcal{F}^1_l(\bar{W}_l) - R_l),$$

by applying a single FAS $V(\nu_{\mathrm{pre}}, \nu_{\mathrm{post}})$-cycle on level $l-1$. Let us denote the resulting approximate solution $(W_{l-1})_{V(\nu_{\mathrm{pre}},\nu_{\mathrm{post}})}(1, R_{l-1}, I^{l-1}_l(\bar{W}_l))$ by $\bar{W}_{l-1}$.

(iii) Improve the solution on Ω_l by first correcting the approximate solution $\bar{W}_l$ obtained in step (i):

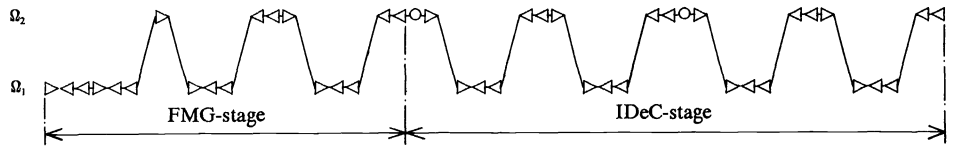
(2.8) $$\bar{W}_l := \bar{W}_l + I_{l-1}^{l}(\Delta W_{l-1}),$$

where $\Delta W_{l-1} = \bar{W}_{l-1} - I_l^{l-1}\bar{W}_l$ is the result of the coarse-grid correction step (ii). Furthermore, improve $\bar{W}_l$ by applying ν_{post} RK4 steps to (2.6): $W_l := (W_l)_{\text{RK4}}(\nu_{\text{post}}, R_l, \bar{W}_l)$.

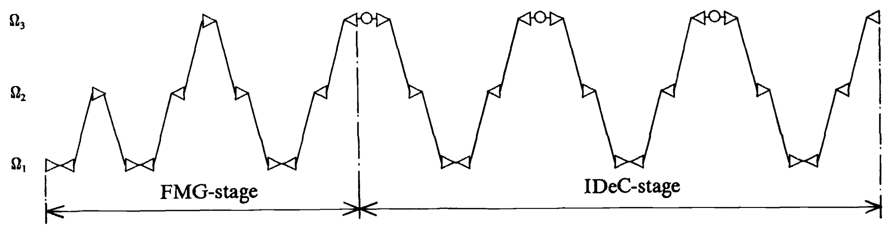
In the FMG-stage in step (i), we have on each starting (locally finest) grid Ω_l, $1 \leqq l \leqq L$: $R_l = 0$. Hence, the initial solution for IDeC as obtained by FMG is at most first-order accurate. In the IDeC stage the starting grid is always the globally finest grid Ω_L, and there we have: $R_L = \mathscr{F}_L^1(W_L) - \mathscr{F}_L^+(W_L)$. This higher-order right-hand side is kept frozen during ν_{FAS} FAS $V(\nu_{\text{pre}}, \nu_{\text{post}})$-cycles per IDeC cycle. Note that with the novel method, we evaluate the higher-order operator at most once per FAS $V(\nu_{\text{pre}}, \nu_{\text{post}})$-cycle, instead of $4 \times (\nu_{\text{pre}} + \nu_{\text{post}}) + 1$ times per FAS $V(\nu_{\text{pre}}, \nu_{\text{post}})$-cycle with the existing method.

In step (ii), note that in the RK4 scheme, the complete right-hand side R_{l-1} is kept frozen. Just as the prolongation operator I_{l-1}^{l}, the restriction operator I_l^{l-1} is such that it also exactly obeys the conservation of cell-integrated mass, momentum, and energy. The restriction operator $\tilde{I}_l^{l-1}$ restricts the defect in the standard way; by summation of mass, momentum, and energy defects over fine-grid cells whose union is a coarse cell. On the coarsest grid (Ω_1), step (ii) (the coarse-grid correction step) is skipped of course.

To illustrate the structure of the complete novel solution method, we give two examples of a complete higher-order solution schedule in Fig. 3. The schedule in Fig. 3(a) is fixed by $L = 2$, $\nu_{\text{pre}} = 1$, $\nu_{\text{post}} = 2$, $\nu_{\text{FAS}} = 2$, $n_{\text{IDeC}} = 2$. The schedule in Fig. 3(b) is fixed by $L = 3$, $\nu_{\text{pre}} = \nu_{\text{post}} = 1$, $\nu_{\text{FAS}} = 1$, $n_{\text{IDeC}} = 3$. In both figures, the marker $\rhd$ denotes a single RK4 step (over Ω_l) preceding a coarse-grid correction, whereas the marker $\lhd$ denotes a single RK4 step after the coarse-grid correction. The marker $\bigcirc$ denotes the computation of $R_L = \mathscr{F}_L^1(W_L) - \mathscr{F}_L^+(W_L)$. Note that the corresponding first-order variants of both schedules (i.e., the variants without any marker $\bigcirc$) are simply obtained by taking $n_{\text{IDeC}} = 0$.



(a)



(b)

FIG. 3. *Examples of complete solution schedule.* (a) $L = 2$, $\nu_{\text{pre}} = 1$, $\nu_{\text{post}} = 2$, $\nu_{\text{FAS}} = 2$, $n_{\text{IDeC}} = 2$. (b) $L = 3$, $\nu_{\text{pre}} = \nu_{\text{post}} = 1$, $\nu_{\text{FAS}} = 1$, $n_{\text{IDeC}} = 3$.

2.3. Analysis. To analyze the convergence properties of the IDeC method proposed, we consider the unsteady, linear, scalar, one-dimensional model equation

$$\frac{\partial w}{\partial t}+c\frac{\partial w}{\partial x}=0, \qquad c>0. \tag{2.9}$$

For the spatial discretization, we consider a grid with a uniformly constant mesh size h. Then we have, for fixed time t, for the first-order upwind discretization,

$$\frac{\partial w}{\partial x}=\frac{-w_{i-1}+w_i}{h}+O(h). \tag{2.10}$$

As a higher-order upwind discretization, we take the Fromm scheme (i.e., van Leer's κ-scheme [19] with $\kappa=0$), which leads to

$$\frac{\partial w}{\partial x}=\frac{w_{i-2}-5w_{i-1}+3w_i+w_{i+1}}{4h}+O(h^2). \tag{2.11}$$

For these two spatial discretizations and the existing explicit solution method, the following Runge-Kutta coefficients can be derived by maximizing the maximally allowable CFL number: $\alpha_1=0.11$, $\alpha_2=0.2767$, $\alpha_3=0.5$ (see [17]). Consistency requires $\alpha_4=1$. The optimization can be redone for our new solution method. However, in the next section we will show that if we simply omit the optimization, the new method already yields both better stability and better smoothing with the α_k's given above (i.e., the α_k's found for the existing higher-order method).

2.3.1. Stability analysis. First we will perform a stability analysis for IDeC with an explicit RK4 scheme as the inner solution method. Let us denote the steady-state analogue of (2.9) by

$$Aw=c\frac{dw}{dx}. \tag{2.12}$$

Assuming that we have periodic boundary conditions, we denote by A_1 and A_+ the first-order and higher-order accurate, linear operators, approximating A. Then, the IDeC process to solve the discrete linear system

$$A_+W=0 \tag{2.13}$$

can be written as

$$A_1W^0=0, \tag{2.14a}$$

$$A_1W^n=(A_1-A_+)W^{n-1}, \qquad n=1,2,\ldots,n_{\text{IDeC}}, \tag{2.14b}$$

where A_1 denotes the matrix resulting from the first-order discretization. Assuming A_1 to be invertible, the corresponding amplification matrix M reads

$$M=I-A_1^{-1}A_+. \tag{2.15}$$

From (2.15) it is clear that to have convergence of IDeC, the spectral radius of M should be smaller than one. It is also clear that the better the resemblance between A_1 and A_+, the faster the convergence of IDeC. (Also important for good efficiency of IDeC is, of course, that A_1 can be inverted in an efficient way.)

Instead of solving each W^n from (2.14b) exactly, we approximate it by means of the explicit RK4 scheme (2.1) with

$$W^{0,0}=W^{n-1}, \tag{2.16a}$$

$$R^{\nu,k-1}=-(A_1W^{\nu,k-1}-(A_1-A_+)W^{0,0}). \tag{2.16b}$$

With ν the number of RK4 steps which allows us to define an (approximate) solution W^n, we now investigate how the corresponding amplification matrix M_ν is related to the amplification matrix M, which corresponds with the exact solution of W^n from (2.14b). If we consider the approximate solution $W^{\nu,4}$, for $\nu = 1$ we have

(2.17) $$W^{1,4} = (P + Q(A_1 - A_+))\,W^{n-1},$$

with

(2.18a) $$P = P(-\Delta t A_1),$$

(2.18b) $$Q = I - A_1^{-1}P,$$

(2.18c) $$P(Z) = I + Z + \alpha_3 Z^2 + \alpha_3\alpha_2 Z^3 + \alpha_3\alpha_2\alpha_1 Z^4.$$

We can easily verify that $A_1 Q = QA_1$, and that $A_1^{-1}P = PA_1^{-1}$. Hence, for M_1 we can write

(2.19) $$M_1 = P + Q(A_1 - A_+).$$

For $\nu > 1$, we can easily find the recurrence relation

(2.20) $$M_\nu = PM_{\nu-1} + Q(A_1 - A_+),$$

which leads to

(2.21) $$M_\nu = (I + P + P^2 + \cdots + P^{\nu-1})Q(A_1 - A_+) + P^\nu.$$

THEOREM. *Let* $\|\cdot\|$ *be some matrix norm. If* $\|P\| < 1$ *and if* P *is invertible, then* $\lim_{\nu\to\infty} M_\nu = M$.

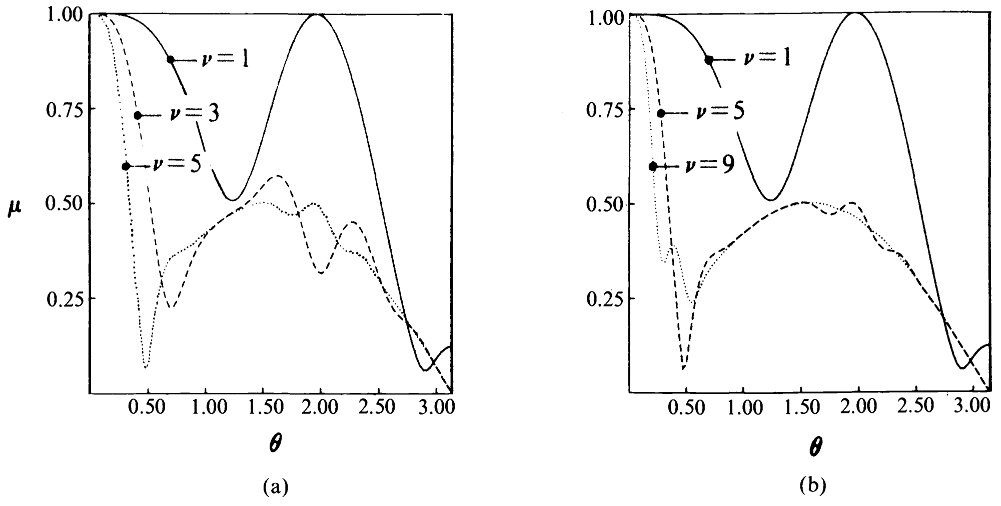
Proof. For all $\nu > 1$, $I + P + P^2 + \cdots + P^{\nu-1} = (I - P)^{-1}(I - P^\nu)$. If $\|P\| < 1$, then $\lim_{\nu\to\infty}(I-P)^{-1}(I - P^\nu) = (I - P)^{-1}$. Hence,

$$\lim_{\nu\to\infty} M_\nu = (I - P)^{-1}Q(A_1 - A_+) = (A_1 Q)^{-1}Q(A_1 - A_+)$$
$$= A_1^{-1}Q^{-1}Q(A_1 - A_+) = M. \qquad \square$$

Local mode analysis, applied to (2.14b) with A_1 and A_+ according to (2.10) and (2.11), respectively, yields for the maximally allowable value of $\sigma \equiv c\Delta t/\Delta x$, $\sigma_{\nu=1} = 2.21$ and $\sigma_{\lim\, \nu\to\infty} = 2.12$. Note that the difference between both values is very small. For an arbitrary ν it is safe and still efficient to take $\sigma = 2.12$. The value $\sigma = 2.12$ is lower than that for the existing method applied to the first-order upwind system ($\sigma = 2.5105$), but higher than that for the existing method applied to the higher-order system ($\sigma = 1.9186$).

For $\sigma = 2.12$ and for increasing ν, Fig. 4 shows the behavior of the convergence factor μ, versus the frequency θ in the range $[0, \pi]$, for the new higher-order method. Already for $\nu = 1$, it appears that the convergence behavior of the new higher-order method is better than that of the existing higher-order method (Fig. 5). Clearly visible for increasing ν is the rapid improvement of the smoothing (i.e., the convergence in the range $\theta \in [\pi/2, \pi]$) and the tendency towards coincidence of the curves. The curves converge to the one that corresponds with the exact solution of (2.14b): $\mu = \frac{1}{2}\sin\theta$, $\theta \in [0, \pi]$. In the next section we will further investigate the smoothing properties of IDeC.

2.3.2. Smoothing analysis. Local mode analysis yields that with the new method, optimal smoothing of the highest frequency, $\theta = \pi$, is obtained for $\sigma = 1.8921$ and $\sigma = 1.4869$. Just as for $\sigma = 2.12$ (Fig. 4), Figs. 6 and 7 give the convergence behavior for $\sigma = 1.8921$ and $\sigma = 1.4869$ with increasing ν. For both values of σ the smoothing is clearly better than for $\sigma = 2.12$. For $\sigma = 1.4869$ it is best.



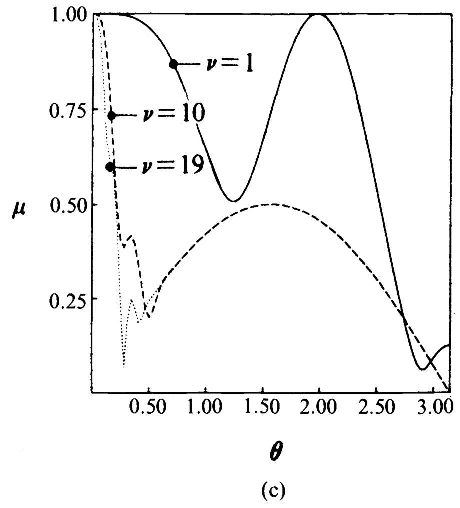


FIG. 4. *Convergence behavior novel higher-order method, for* $\sigma = 2.12$. (a) $\nu = 1, 3, 5$. (b) $\nu = 1, 5, 9$. (c) $\nu = 1, 10, 19$.

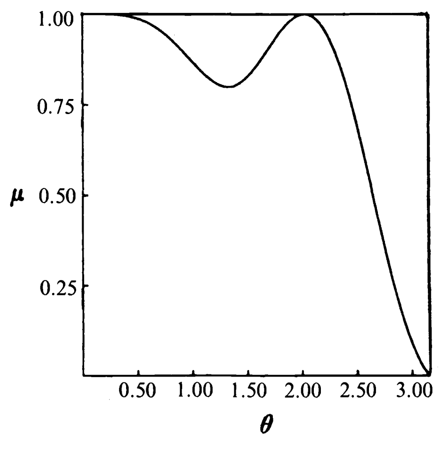


FIG. 5. *Convergence behavior existing higher-order method, for* $\sigma = 1.9186$.

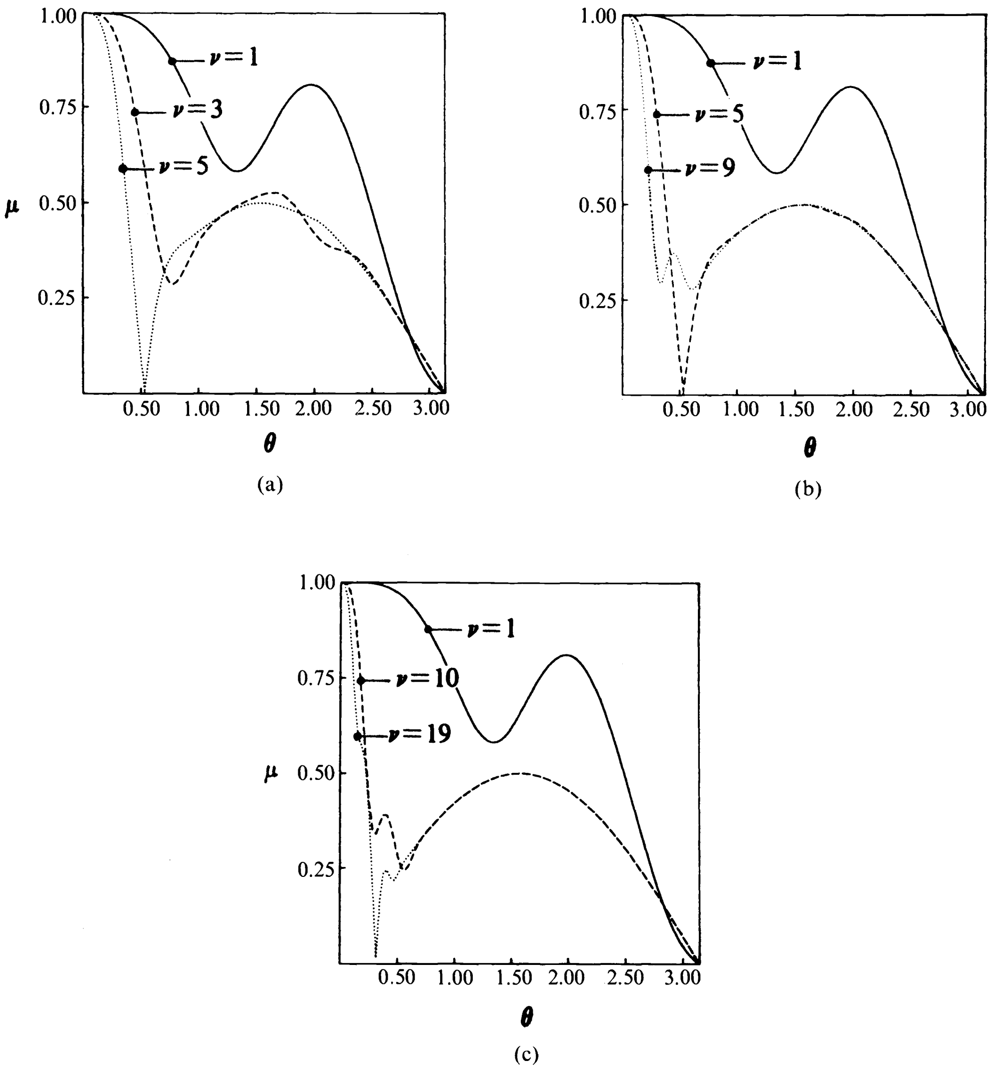


FIG. 6. *Convergence behavior novel higher-order method, for* $\sigma = 1.8921$. (a) $\nu = 1, 3, 5$. (b) $\nu = 1, 5, 9$. (c) $\nu = 1, 10, 19$.

In Figs. 8, 9, and 10 we show the smoothing behavior for varying σ, the θ-range considered being $[\pi/2, \pi]$, and the quantity μ along the vertical axis being the maximum smoothing factor found over this range. We consider successively: the first-order method (Fig. 8), the existing higher-order method (Fig. 9), and the new higher-order method (Fig. 10). When we compare the results of the existing higher-order method and the new higher-order method (Figs. 9 and 10), we find that the new method clearly has better smoothing properties. The new method appears to have even better smoothing properties than the first-order method (compare Figs. 8 and 10). Note in particular that the σ-range over which its smoothing is good is much wider.

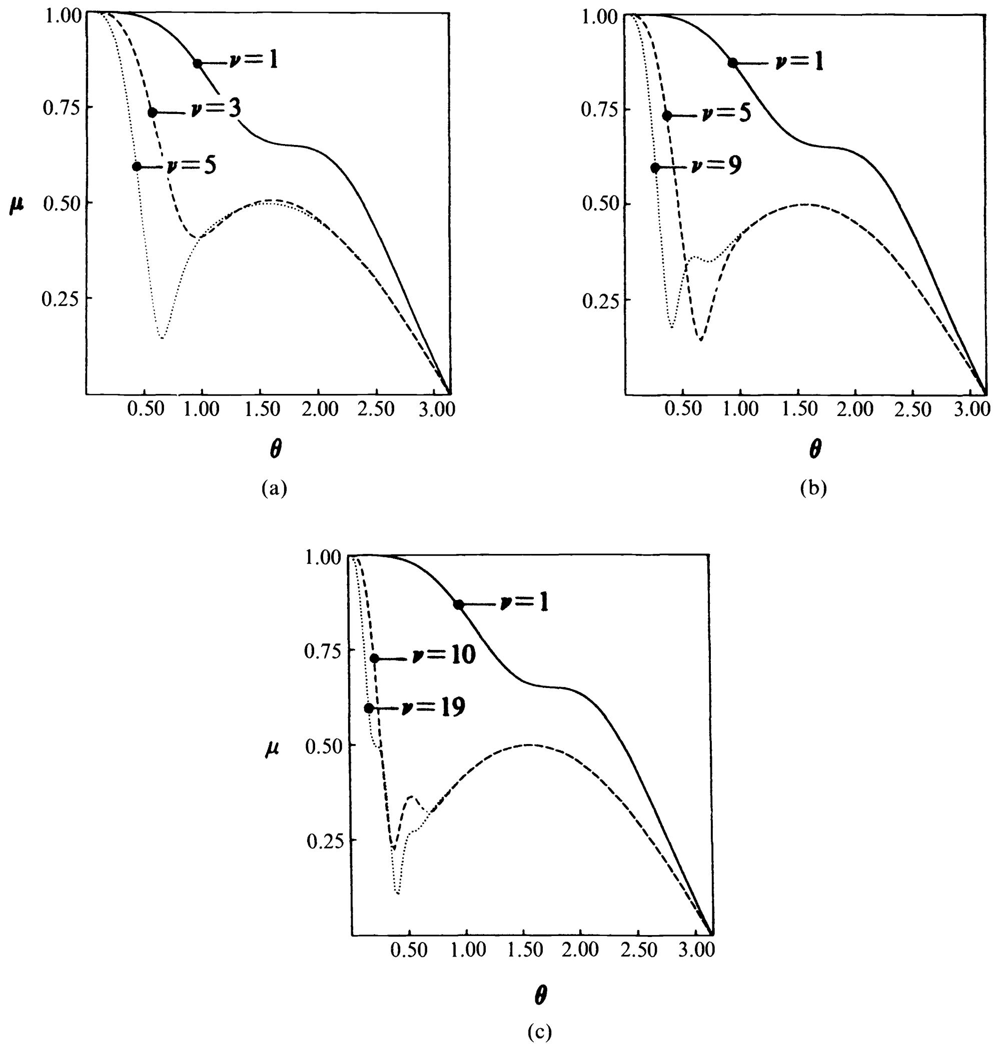


FIG. 7. *Convergence behavior novel higher-order method, for* $\sigma = 1.4869$. (a) $\nu = 1, 3, 5$. (b) $\nu = 1, 5, 9$. (c) $\nu = 1, 10, 19$.

3. Numerical results. In order to verify the previously predicted better stability and convergence properties of the novel higher-order method, we compute the standard transonic channel flow from [21] with the two-dimensional Euler equations. Three finest grids are considered: a 161-vertices grid (Fig. 11), a 585-vertices grid that is about twice as fine (see [16]), and a 2225-vertices grid that is about four times as fine. The corresponding solution schedules applied are a four-, five, and six-level schedule ($L = 4, 5, 6$), respectively, all with $\nu_{\text{pre}} = \nu_{\text{post}} = 1$, for all l.

In Figs. 12(a)–12(c) we present various convergence histories as obtained for $L = 4, 5, 6$, respectively. The convergence results presented are those of (i) the first-order discretized Euler equations solved by means of the nonlinear multigrid iteration (dotted lines), and those of higher-order discretized Euler equations solved by means of (ii) the existing higher-order method (dashed lines), and (iii) the novel higher-order method

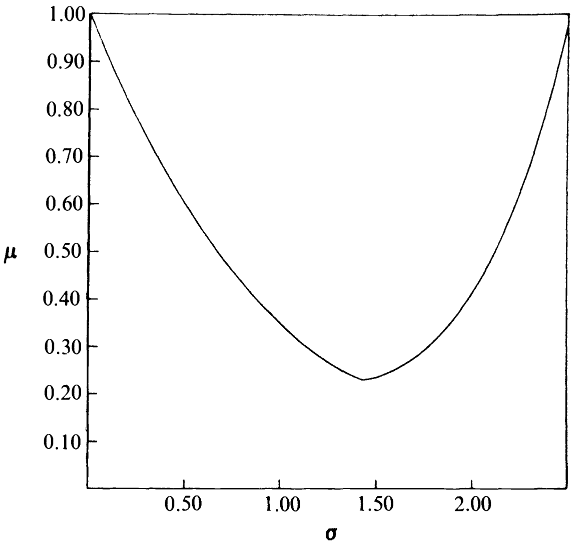


FIG 8. *Smoothing behavior for varying σ, first-order method.*

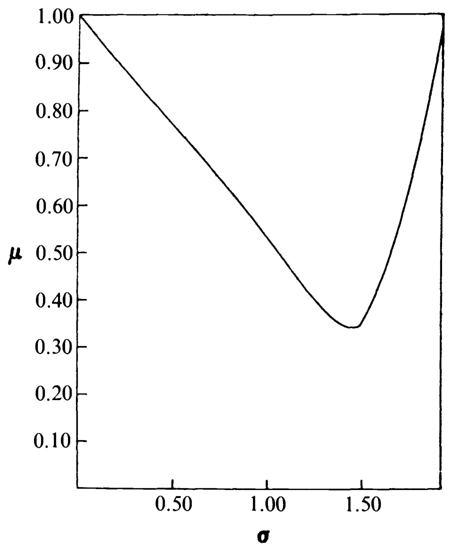


FIG 9. *Smoothing behavior for varying σ, existing higher-order method.*

(solid lines). In all three graphs, the residual considered is the L_2-norm of the error in the conservation of mass over all the finest-grid cells. Furthermore, in all three graphs the number of cycles indicated along the horizontal axis is (i) the number of FAS cycles in case of both the first-order method and the existing higher-order method, and (ii) the number of IDeC cycles in case of the new higher-order method. Note that with the new higher-order method, for $\nu_{\text{FAS}} = 2, 5, 10$ the number of inner FAS cycles is, respectively, 2, 5, and 10 times larger than the number of indicated IDeC cycles. (Only for $\nu_{\text{FAS}} = 1$ does the number of FAS cycles equal the number of IDeC cycles.)

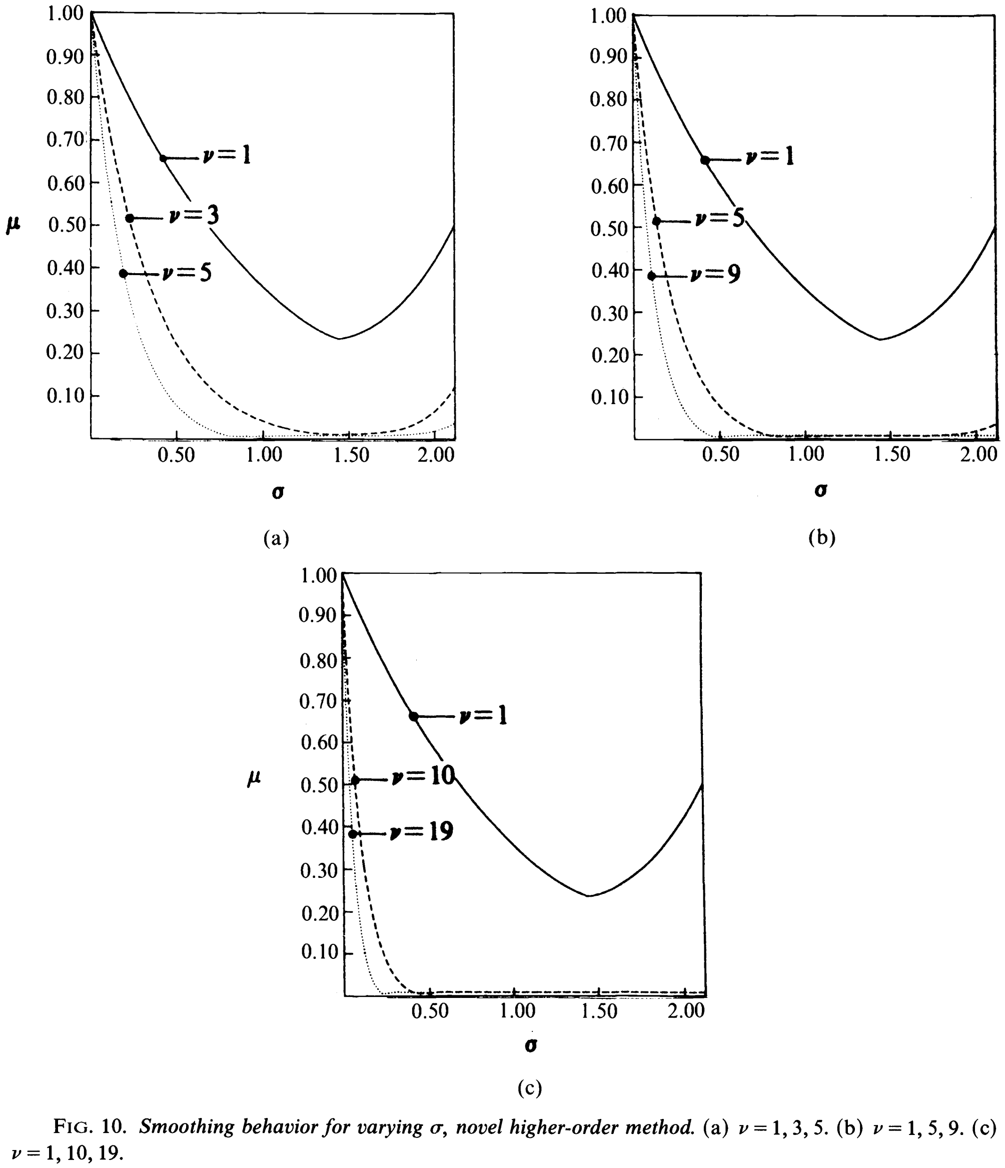


FIG. 10. *Smoothing behavior for varying σ, novel higher-order method.* (a) $\nu = 1, 3, 5$. (b) $\nu = 1, 5, 9$. (c) $\nu = 1, 10, 19$.

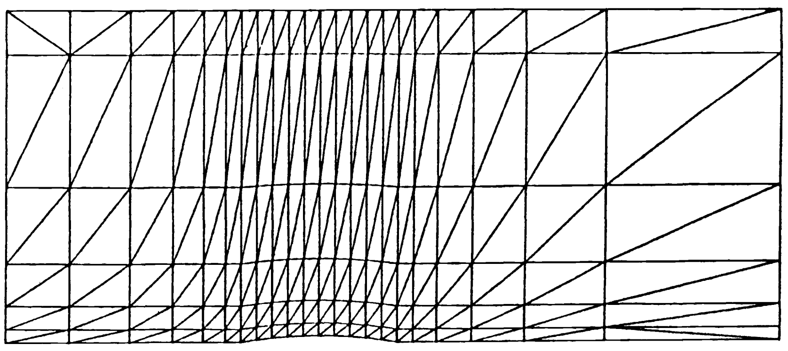
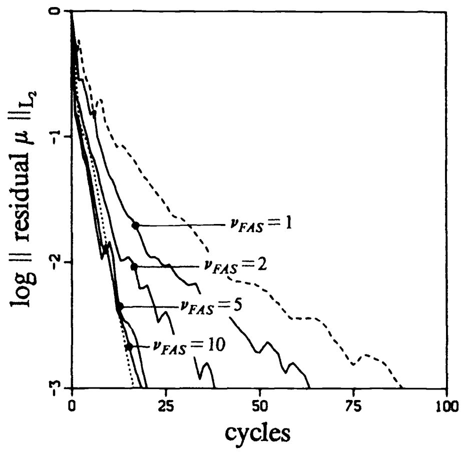
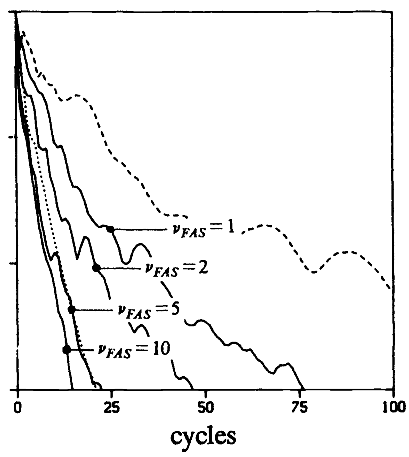


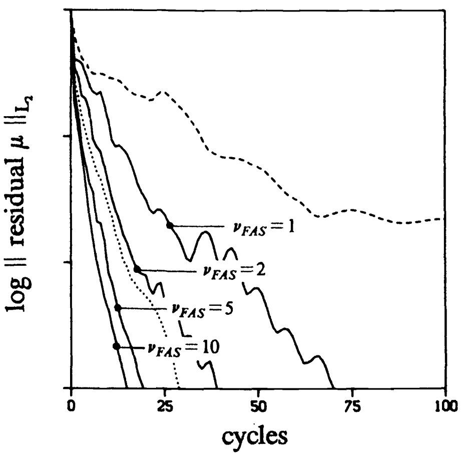
FIG. 11. *Channel from* [20], *with* 161-*vertices grid.*



(a)



(b)

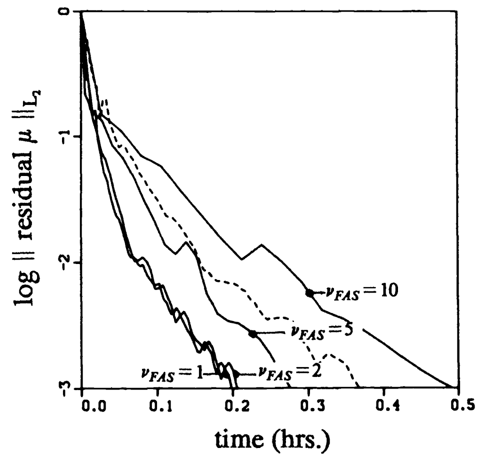


(c)

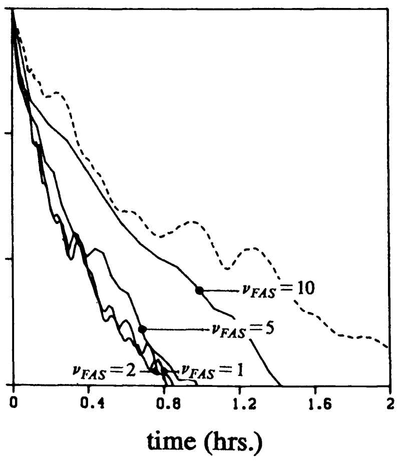
FIG. 12. *Convergence histories*: *first-order method*, $\cdots\cdot$; *existing higher-order method*, ----; *novel higher-order method*, ——. (a) 161-*vertices grid* ($L=4$). (b) 585-*vertices grid* ($L=5$). (c) 2225-*vertices grid* ($L=6$).

All convergence histories start at the end of the FMG stage (Fig. 3). In agreement with the theoretical results found in § 2.3, for all four values of ν_{FAS} (so also for $\nu_{\text{FAS}}=1$), the new method does indeed give a better convergence than the existing higher-order method. For decreasing mesh width, the convergence of the new higher-order method becomes even relatively better than that of the first-order method. (For all four values of ν_{FAS} under consideration, the corresponding convergence histories in Fig. 12 show a better grid-independency than those of the multigrid method applied to the first-order discretized equations.) This better performance is probably due to better smoothing in the new method. (In § 2.3, by model analysis, we have found that the new method has better smoothing properties than the first-order method.)

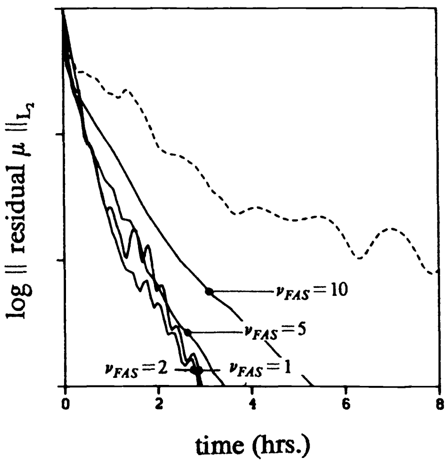
As for the actual order of accuracy, if we took the converged higher-order accurate solution obtained on the 2225-vertices grid as the reference solution, we measured local orders of accuracy in the range $[O(h^{1.4}), O(h^{2.3})]$ for the solutions on the coarser



(a)



(b)



(c)

FIG. 13. *Efficiency histories: existing higher-order method, - - - -; novel higher-order method, ——.* (a) 161-*vertices grid* ($L=4$). (b) 585-*vertices grid* ($L=5$). (c) 2225-*vertices grid* ($L=6$).

grids (the 585-vertices grid and the 161-vertices grid). The global order of accuracy appears to be almost $O(h^2)$.

Finally, the important question of which of the various higher-order methods is the most efficient still remains. To answer this question, we give the higher-order efficiency histories in Figs. 13(a)–13(c). The indicated computing times have been obtained on a Sequent. (No efforts have been undertaken to make efficient use of the parallelization features of the machine. What interests us here is the relative efficiency of the higher-order methods only.) Since the sizes of the three grids considered are related to each other by approximately a factor 4, we have related the scales along the horizontal axes accordingly. Concerning the relative efficiency of the novel higher-order method, for the four values of ν_{FAS} considered, it appears that for all three grids the best efficiency is obtained with $\nu_{\text{FAS}}=1$ (so just as in [14], for the schedule with only a single FAS cycle per IDeC cycle). Furthermore, it is significant that the novel method with $\nu_{\mathrm{FAS}} = 1$ appears to be more efficient than the existing higher-order method. Due to the better grid-independency of the novel method, this relatively better efficiency becomes even increasingly better with decreasing meshwidth.

4. Conclusions. Fully implicit solution methods for higher-order discretized equations may strongly benefit from iterative defect correction when these systems of discretized equations are not easily invertible, which is often the case with higher-order accurate discretizations. Fully explicit solution methods may also profit from iterative defect correction. Here the profits are faster convergence and higher efficiency. The defect correction method appears to lead to greater stability (and hence to greater robustness) than the existing (standard) explicit method. Compared to the existing explicit method, it possesses remarkably good smoothing properties, in fact even better than the first-order method. Last but not least, its convergence rate appears to be grid-independent. For upwind discretizations, the "price" which has to be paid for using defect correction iteration, a slightly more complex algorithm, is negligible, because of the direct availability of an appropriate approximate operator; the first-order upwind operator.

Acknowledgments. The authors want to thank Joke Sterringa, P. Wesseling, and the referees for their suggestions in improving this paper.

REFERENCES

[1] K. Baba and M. Tabata, *On a conservative upwind finite element scheme for convective diffusion equations*, RAIRO Numer. Anal., 15 (1981), pp. 3-25.

[2] K. Böhmer, P. W. Hemker, and H. J. Stetter, *The defect correction approach*, Comput. Suppl., 5 (1984), pp. 1-32.

[3] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comput., 31 (1977), pp. 330-399.

[4] ———, *Guide to multigrid development*, Lecture Notes in Math. 960, Springer-Verlag, Berlin, 1982, pp. 220-312.

[5] A. Dervieux, J.-A. Desideri, F. Fezoui, B. Palmerio, J. P. Rosenblum, and B. Stoufflet, *Euler calculations by upwind finite element methods and adaptive mesh algorithms*, Notes on Numerical Fluid Mechanics, No. 26, Vieweg, Braunschweig, 1989, pp. 138-156.

[6] J.-A. Desideri and P. W. Hemker, *Analysis of the convergence of iterative implicit and defect-correction algorithms for higher-order problems*, Rapport de Recherche 1200, INRIA Sophia-Antipolis, Valbonne, 1990.

[7] L. Fezoui and B. Stoufflet, *A class of implicit upwind schemes for Euler simulations with unstructured grids*, J. Comput. Phys., 84 (1989), pp. 174-206.

[8] S. K. Godunov, *Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics*, Mat. Sb., 47 (1959), pp. 271-306. (Cornell Aeronautical Lab. Transl. from Russian.)

[9] W. Hackbusch, *Bemerkungen zur iterierten Defektkorrektur und zu ihrer Kombination mit Mehrgitterverfahren*, Rev. Roumaine Math. Pures Appl., 26 (1981), pp. 1319-1329.

[10] ———, *Multi-grid methods and applications*, Springer-Verlag, Berlin, 1985.

[11] P. W. Hemker, *Mixed defect correction iteration for the solution of a singular perturbation problem*, Comput. Suppl., 5 (1984), pp. 123-145.

[12] ———, *Defect correction and higher order schemes for the multi grid solution of the steady Euler equations*, Lecture Notes in Math. 1228, Springer-Verlag, Berlin, 1986, pp. 149-165.

[13] A. Jameson, *Solution of the Euler equations for two dimensional transonic flow by a multigrid method*, Appl. Math. Comput., 13 (1983), pp. 327-355.

[14] B. Koren, *Defect correction and multigrid for an efficient and accurate computation of airfoil flows*, J. Comput. Phys., 77 (1988), pp. 183-206.

[15] ———, *Multigrid and defect correction for the steady Navier-Stokes equations*, J. Comput. Phys., 87 (1990), pp. 25-46.

[16] M.-H. LALLEMAND AND A. DERVIEUX, *A multigrid finite element method for solving the two-dimensional Euler equations*, Lecture Notes in Pure and Appl. Math. 110, Marcel Dekker, New York, 1988, pp. 337-363.

[17] M.-H. LALLEMAND, *Dissipative properties of Runge-Kutta schemes with upwind spatial approximation for the Euler equations*, Rapport de Recherche 1179, INRIA Sophia-Antipolis, Valbonne, 1990.

[18] B. VAN LEER, *Towards the ultimate conservative difference scheme* V. *A second-order sequel to Godunov's method*, J. Comput. Phys., 32 (1979), pp. 101-136.

[19] ———, *Upwind-difference methods for aerodynamic problems governed by the Euler equations*, Lectures in Appl. Math. 22, Part 2, Amer. Math. Soc., Providence, RI, 1985, pp. 327-336.

[20] S. OSHER AND F. SOLOMON, *Upwind-difference schemes for hyperbolic systems of conservation laws*, Math. Comput., 38 (1982), pp. 339-374.

[21] A. RIZZI AND H. VIVIAND, *Numerical methods for the computation of inviscid transonic flows with shock waves*, Notes on Numerical Fluid Mechanics, No. 3, Vieweg, Braunschweig, 1981.

[22] P. L. ROE, *Approximate Riemann solvers, parameter vectors and difference schemes*, J. Comput. Phys., 43 (1981), pp. 357-372.

[23] P. ROSTAND AND B. STOUFFLET, TVD *schemes to compute compressible viscous flows on unstructured meshes*, Notes on Numerical Fluid Mechanics, No. 24, Vieweg, Braunschweig, 1989, pp. 510-520.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 971–1006, July 1993 012

THE ASYMPTOTIC SPECTRA OF BANDED TOEPLITZ AND QUASI-TOEPLITZ MATRICES*

RICHARD M. BEAM[†] AND ROBERT F. WARMING[†]

Abstract. Toeplitz matrices occur in many mathematical as well as scientific and engineering investigations. This paper considers the spectra of *banded* Toeplitz and *quasi*-Toeplitz matrices with emphasis on nonnormal matrices of arbitrarily large order and relatively small bandwidth. These are the type of matrices that appear in the investigation of stability and convergence of difference approximations to partial differential equations. Quasi-Toeplitz matrices are the result of non-Dirichlet boundary conditions for the difference approximations. The eigenvalue problem for a banded Toeplitz or quasi-Toeplitz matrix of large order is, in general, analytically intractable and (for nonnormal matrices) numerically unreliable. An asymptotic (matrix order tends to infinity) approach partitions the (asymptotic) spectrum of a quasi-Toeplitz matrix into two parts; namely, the analysis for the boundary condition *independent* spectrum and the analysis for the boundary condition *dependent* spectrum. The boundary condition independent spectrum is the same as the *pure* Toeplitz matrix spectrum. Algorithms for computing both parts of the asymptotic spectrum are presented. Examples are used to demonstrate the utility of the algorithms, to present some interesting spectra, and to point out some of the numerical difficulties encountered when conventional matrix eigenvalue routines are employed for nonnormal matrices of large order. The analysis for the Toeplitz spectrum also leads to a diagonal similarity transformation that improves conventional numerical eigenvalue computations. Finally, the algorithm for the asymptotic spectrum is extended to the Toeplitz generalized eigenvalue problem which occurs, for example, in the stability analysis of Padé type difference approximations to differential equations.

Key words. Toeplitz matrices, eigenvalues, spectrum, stability

AMS subject classifications. 65F15, 65M10, 76N10

1. Introduction.

A Toeplitz matrix has the property that the entries are constant along diagonals parallel to the main diagonal. If we define the sequence

$$a_{-p}, a_{-p+1}, \cdots, a_0, \cdots, a_{q-1}, a_q; \quad \text{where} \quad a_{-p}, a_q \neq 0 \tag{1.1}$$

and p and q are specified positive integers, then the elements of a *banded* square Toeplitz matrix of order J and bandwidth $p+q+1$ are given by

$$a_{ij} = a_{j-i} \tag{1.2}$$

if a_{j-i} is a member of the sequence (1.1) and zero otherwise. Since the eigenvalue analysis for triangular matrices is trivial, we have restricted our attention to *nontriangular* matrices ($p, q > 0$). For example, if we choose $p = 1$ and $q = 2$ and denote

*Received by the editors October 25, 1991; accepted for publication (in revised form) August 28, 1992. The main results of this paper were presented at the 14th Biennial Conference on Numerical Analysis, Dundee, Scotland, June 25-28, 1991.

[†]Computational Fluid Dynamics Branch, NASA Ames Research Center, Moffett Field, California 94035

the banded Toeplitz matrix by $\mathbf{A}$ then

$$(1.3)\qquad \mathbf{A} = \begin{bmatrix} a_0 & a_1 & a_2 & & & & & & \\ a_{-1} & a_0 & a_1 & a_2 & & & & & \\ & a_{-1} & a_0 & a_1 & a_2 & & O & & \\ & & \cdot & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \cdot & \\ & & O & & & a_{-1} & a_0 & a_1 & a_2 \\ & & & & & & a_{-1} & a_0 & a_1 \\ & & & & & & & a_{-1} & a_0 \end{bmatrix},$$

where the bandwidth is four and the matrix order J is arbitrary. Our interest is in the spectra, i.e., the eigenvalues, for banded Toeplitz matrices of arbitrarily large order, i.e, $J \to \infty$.

A banded *quasi*-Toeplitz matrix is defined to be a banded Toeplitz matrix where there are a limited number of row changes constrained as follows: There are at most p altered rows among the first p rows and at most q altered rows among the last q rows. Since p and q are fixed and J is assumed to be large, there are only a relatively few altered rows. Here we use *quasi* in the conventional sense meaning *almost* or *nearly.* For example, a quasi-Toeplitz cousin of the Toeplitz matrix (1.3) is

$$(1.4)\qquad \mathbf{A} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & & & & & \\ a_{-1} & a_0 & a_1 & a_2 & & & & & \\ & a_{-1} & a_0 & a_1 & a_2 & & O & & \\ & & \cdot & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \cdot & \\ & & O & & & a_{-1} & a_0 & a_1 & a_2 \\ & & & & & c_{24} & c_{23} & c_{22} & c_{21} \\ & & & & & c_{14} & c_{13} & c_{12} & c_{11} \end{bmatrix}.$$

(The advantage of the unusual indexing for the c entries of $\mathbf{A}$ will become apparent in §4.2). We assume that the modified elements are constants and therefore independent of J. The bandwidth of a quasi-Toeplitz matrix may be larger than the bandwidth of its pure Toeplitz cousin, e.g., the quasi-Toeplitz matrix defined by (1.4) has a larger bandwidth than its Toeplitz cousin defined by (1.3).

Another important relative of a banded Toeplitz matrix is its *circulant* cousin. Each row of a circulant matrix is constructed by cycling the previous row forward one element. For example, if the first row of a matrix of order J is

$$(1.5)\qquad [a_0, a_1, \cdots, a_q, 0, \cdots, 0, a_{-p}, \cdots, a_{-2}, a_{-1}],$$

this process defines the circulant cousin of the Toeplitz matrix defined by (1.2). In particular, the circulant cousin of the Toeplitz matrix (1.3) is

$$
\mathbf{A} = \begin{bmatrix}
a_0 & a_1 & a_2 & & & & & a_{-1} \\
a_{-1} & a_0 & a_1 & a_2 & & & & \\
& a_{-1} & a_0 & a_1 & a_2 & & O & \\
& & \cdot & \cdot & \cdot & \cdot & & \\
& & & \cdot & \cdot & \cdot & \cdot & \\
& & & & \cdot & \cdot & \cdot & \cdot \\
& & O & & a_{-1} & a_0 & a_1 & a_2 \\
a_2 & & & & & a_{-1} & a_0 & a_1 \\
a_1 & a_2 & & & & & a_{-1} & a_0
\end{bmatrix}. \tag{1.6}
$$

We will also refer to the matrix defined by (1.5) as the circulant cousin of a quasi-Toeplitz matrix, e.g., (1.6) is the circulant cousin of (1.4). A circulant matrix is also Toeplitz because the entries are constant along diagonals. The eigenvalues (and eigenvectors) of any circulant matrix can be determined analytically [1]. The spectrum for a circulant banded Toeplitz matrix is a subset of the spectrum for the doubly infinite (order) banded Toeplitz matrix [9].

Toeplitz matrices are important in mathematics as well as scientific and engineering applications [5], [4]. Our interest [15] arose from an investigation of stability and convergence of difference approximations to initial-boundary-value problems for partial differential equations. The analysis of these difference approximations leads to a study of the spectra for banded quasi-Toeplitz matrices of large order. The element changes from the pure Toeplitz matrix are the result of applying non-Dirichlet boundary conditions to the difference equations. The bandwidth of the matrix is determined by the width of the difference stencil and is small compared with the order of the matrix. For example, if the method of lines is applied to a scalar hyperbolic partial differential equation and if a four-point spatial difference approximation is used to approximate the spatial derivative, one obtains the quasi-Toeplitz matrix

$$
\begin{bmatrix}
-\frac{2\alpha+3}{2} & 3\alpha+2 & -\frac{6\alpha+1}{2} & \alpha & & & & \\
-\frac{1}{3} & -\frac{1}{2} & 1 & -\frac{1}{6} & & & & \\
& -\frac{1}{3} & -\frac{1}{2} & 1 & -\frac{1}{6} & & O & \\
& & \cdot & \cdot & \cdot & \cdot & & \\
& & & \cdot & \cdot & \cdot & \cdot & \\
& & & & \cdot & \cdot & \cdot & \cdot \\
& & O & & -\frac{1}{3} & -\frac{1}{2} & 1 & -\frac{1}{6} \\
& & & & & -\frac{1}{3} & -\frac{1}{2} & 1 \\
& & & & & -\beta & \frac{6\beta-1}{2} & -3\beta
\end{bmatrix}, \tag{1.7}
$$

where the parameters α and β are introduced by the numerical boundary conditions [15]. If we choose $\alpha = 6/5$ and $\beta = -4/5$ and calculate the spectrum of (1.7) for a family of increasing matrix orders, we obtain the spectra shown by the open circles in the complex plane in Fig. 1.1. Note the four "isolated" eigenvalues in Fig. 1.1(d). (The abscissa and the ordinate of each eigenvalue, respectively, represent the real part and the imaginary part of the eigenvalue.) In the Lax stability analysis of a difference method for an initial-boundary-value problem, the location of the spectrum in the complex plane determines a necessary (but not sufficient) condition for stability. The GKS (Gustafsson, Kreiss, and Sundström [6]) stability analysis involves checking the Cauchy stability (location of the circulant cousin spectrum) and also checking the

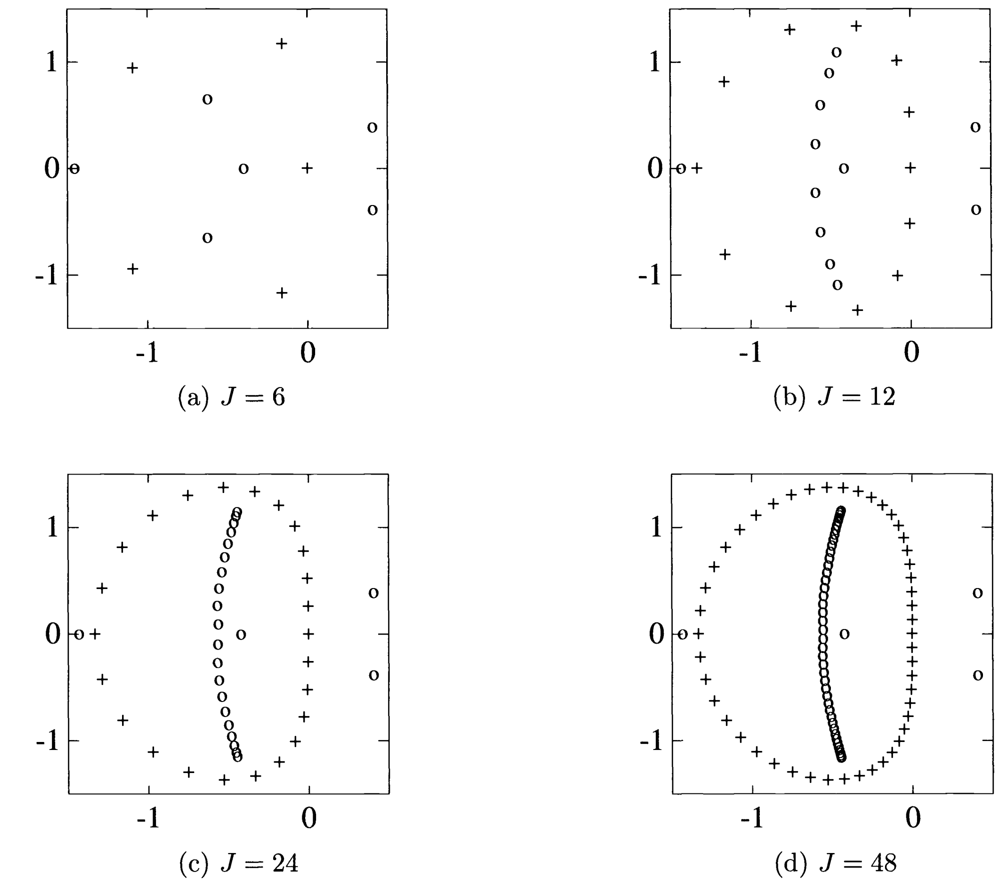


FIG. 1.1. *Spectra for the quasi-Toeplitz matrix* (1.7), *with* $\alpha = 6/5$ *and* $\beta = -4/5$, (o *symbol) and the spectra for the circulant Toeplitz cousin* (+ *symbol) for matrix order* J.

location and asymptotic behavior of "isolated" eigenvalues. The spectrum (spectral radius) also determines the asymptotic iterative convergence rate for fixed J. For the purposes of this paper we are interested in the spectrum for large matrix order, e.g., Fig. 1.1(d). For reference we have also plotted the spectrum (+ symbols) for the circulant cousin of (1.7) in Fig. 1.1.

In §2 the eigenvalue problem for a banded Toeplitz or quasi-Toeplitz matrix is formulated as a difference equation with appropriate boundary conditions. In general, the resulting boundary value problem is analytically intractable. In §3 we assume that the order of a quasi-Toeplitz matrix is large ($J \to \infty$) and use an asymptotic approach to partition the spectrum into two independent sets. The partition is between boundary condition *independent* spectrum and boundary condition *dependent* spectrum. Algorithms for the computation of both parts of the spectrum are described in §4. These algorithms involve the solution of algebraic equations with degree proportional to the bandwidth of the matrix. The algorithm for the boundary condition dependent spectrum closely parallels the eigenvalue analysis of Godunov and Ryabenkii [2], Kreiss

[7] and Gustafsson, Kreiss, and Sundström [6]. Plots of the asymptotic spectra of Toeplitz matrices often exhibit surprising geometrical shapes. Some illustrative examples are shown in §5. The algorithms for computing the spectra are applied to both Toeplitz and quasi-Toeplitz matrices in §6. If conventional numerical eigenvalue algorithms are used to compute the spectra for nonnormal matrices of large order, errors in the spectra may be encountered. In §7 we introduce a simple similarity transformation that improves the accuracy of conventional eigenvalue algorithms for a large class of banded Toeplitz and quasi-Toeplitz matrices. Finally, the algorithm for the asymptotic Toeplitz spectrum is extended to the generalized eigenvalue problem in §8.

2. The eigenvalue problem: a difference equation representation.

Let $\mathbf{A}$ represent a $J \times J$ banded Toeplitz matrix defined by (1.2). The eigenvalue problem for $\mathbf{A}$ is defined by

$$\mathbf{A}\boldsymbol{\phi} = \lambda\boldsymbol{\phi} \tag{2.1}$$

where λ is the eigenvalue and $\boldsymbol{\phi}$ is the eigenvector:

$$\boldsymbol{\phi}^T = [\phi_1, \phi_2, \cdots, \phi_{J-1}, \phi_J]. \tag{2.2}$$

The eigenvalue problem (2.1) is equivalent to the system of homogeneous equations

$$\sum_{m=-p}^{q} a_m \phi_{j+m} = \lambda \phi_j, \qquad j = 1, 2, \cdots, J \tag{2.3}$$

with p homogeneous Dirichlet boundary conditions at the left boundary and q homogeneous boundary conditions at the right boundary

$$\phi_{-m} = 0, \quad m = 0, 1, \cdots, p-1, \tag{2.4a}$$

$$\phi_{J+m} = 0, \quad m = 1, 2, \cdots, q. \tag{2.4b}$$

Note that fictitious points have been introduced in defining the boundary conditions (2.4). It is apparent that (2.3) is the generic or jth equation of the system (2.1). A difference equation representation of the eigenvalue problem, e.g., (2.3) and (2.4), has been used for the purposes of analysis, e.g., [13], and occurs naturally in the stability analysis of finite difference approximations.

Equation (2.3) is a $(p+q)$th order difference equation for ϕ_j, and we look for a solution of the form

$$\phi_j = \kappa^j, \tag{2.5}$$

where κ is a complex constant. Insertion of (2.5) into (2.3) yields

$$\lambda = \sum_{m=-p}^{q} a_m \kappa^m. \tag{2.6}$$

This is an algebraic equation of degree $p+q$ in the unknown κ. If there are $p+q$ distinct κ's, the general solution of (2.3) is

$$\phi_j = \sum_{m=1}^{p+q} \beta_m \kappa_m^j, \tag{2.7a}$$

where the κ_m's are the roots of (2.6) and the β_m's are arbitrary constants. If there are only d ($d < p+q$) distinct κ's with multiplicity $r_1, r_2, \cdots, r_d$, then the general solution of (2.3) is

$$\phi_j = \sum_{m=1}^{d} \sum_{n=0}^{r_m-1} \beta_{mn} j^n \kappa_m^j. \tag{2.7b}$$

The constants β are determined by inserting (2.7) into the boundary conditions (2.4). If the bandwidth is three, i.e., $p=q=1$, the eigenvalue problem for the pure Toeplitz problem can be solved analytically [10], [15]. However, if the bandwidth is greater than three, the problem is analytically intractable.

For a quasi-Toeplitz matrix the difference equation formulation of the eigenvalue problem differs from the pure Toeplitz problem only in the boundary conditions. The difference equation (2.3) is still valid if appropriate extrapolation boundary conditions are generated. These extrapolation boundary conditions depend on the matrix entries that change the matrix from Toeplitz to quasi-Toeplitz. Extrapolation boundary conditions have the general form

$$P_m(E)\phi_{-m} = 0, \quad m = 0, 1, \cdots, p-1, \tag{2.8a}$$

$$Q_m(E)\phi_{J+m} = 0, \quad m = 1, 2, \cdots, q, \tag{2.8b}$$

where $P_m(E)$ and $Q_m(E)$ are polynomial operators and E is the shift operator defined by $E\phi_j = \phi_{j+1}$. The subscripts on the operators indicate that, in general, the operators are different for each value of m. As an example, the extrapolation boundary conditions for the matrix (1.7) are

$$P_0(E)\phi_0 = 0, \tag{2.9a}$$

$$Q_1(E)\phi_{J+1} = 0, \tag{2.9b}$$

$$Q_2(E)\phi_{J+2} = 0, \tag{2.9c}$$

where, after some algebra, one finds

$$P_0(E) = (E-1)^3(3\alpha E - 1), \tag{2.10a}$$

$$Q_1(E) = 1, \tag{2.10b}$$

$$Q_2(E) = (1 - E^{-1})^3(1 + 6\beta E^{-1}). \tag{2.10c}$$

Although the extrapolation boundary conditions are easy to derive, we will find it more convenient to implement an eigenvalue algorithm that uses the boundary difference equations directly. For example, the first row and the last two rows of the matrix (1.4) have entry changes. For the eigenvalue problem (2.1) the difference equation (2.3) is still valid at $j = 2, 3, \cdots, J-2$ and modified difference equations apply at $j = 1, J-1, J$. The general form of difference equations for the *left* boundary is

$$\mathbf{B}\tilde{\boldsymbol{\phi}} = \lambda \mathbf{D}\tilde{\boldsymbol{\phi}}, \tag{2.11}$$

where $\mathbf{B}$ is a rectangular matrix with p rows and r columns and $\tilde{\boldsymbol{\phi}}^T = [\phi_1, \phi_2, \cdots, \phi_r]$. The matrix $\mathbf{D}$ is a $p \times r$ rectangular diagonal matrix with unit entries on the diagonal. Similar equations apply at the right boundary.

The eigenvalue problem for the circulant cousin of (1.2) is (2.3) with periodic boundary conditions

$$\phi_j = \phi_{j+J}. \tag{2.12}$$

It is well known that the eigenvalue problem for any circulant matrix can be solved analytically. By inserting (2.5) into (2.12), one obtains

$$\kappa^J = 1. \tag{2.13}$$

Hence the κ's are the J roots of unity

$$\kappa_\ell = e^{i\theta_\ell}, \quad \text{where} \quad \theta_\ell = 2\ell\pi/J, \quad \ell = 1, 2, \cdots, J. \tag{2.14}$$

The circulant spectrum is obtained by substituting (2.14) into (2.6):

$$\lambda_\ell = \sum_{m=-p}^{q} a_m e^{im\theta_\ell}. \tag{2.15}$$

As an example, the spectrum of the circulant matrix (1.6) with coefficients

$$[a_{-1}, \underline{a_0}, a_1, a_2] = [-1/3,\ \underline{-1/2},\ 1,\ -1/6] \tag{2.16}$$

reduces to the compact form

$$\lambda_\ell = -4w_\ell^2/3 - i[1 + 2w_\ell/3]\sin\theta_\ell, \tag{2.17}$$

where

$$w_\ell = \sin^2(\theta_\ell/2), \quad \theta_\ell = 2\ell\pi/J, \quad \ell = 1, 2, \cdots, J. \tag{2.18}$$

The eigenvalue locus is the oval shaped dashed curve plotted in Fig. 5.1. In equation (2.16) and the remainder of this paper we use the underline to indicate the main diagonal element.

3. Partition of the asymptotic spectrum. Let A_J be a $J \times J$ banded quasi-Toeplitz (or Toeplitz) matrix. Here A_J denotes the Jth member of a family of matrices of arbitrarily large order. In the limit $J \to \infty$ one can partition the asymptotic spectrum into two sets. The partition is between the set which is *independent* of the boundary conditions and the set (possibly empty) which is *dependent* on the boundary conditions.

First we introduce some definitions. The spectrum of A_J is denoted by

$$\sigma_J = \{\lambda | \det(A_J - \lambda I) = 0\}. \tag{3.1}$$

DEFINITION 3.1. The asymptotic spectrum of the family $\{A_J\}$ is the set $\mathcal{S}$ defined by

$$\mathcal{S} = \{\lambda \mid \lambda = \lim_{m\to\infty} \lambda_m, \lambda_m \in \sigma_{i_m}, \lim_{m\to\infty} i_m = \infty\}. \tag{3.2}$$

In the special case where A_J is a pure Toeplitz matrix, we denote the asymptotic spectrum by the set $\mathcal{B}$, i.e.,

$$\mathcal{B} = \mathcal{S} \tag{3.3}$$

to conform with the notation of Schmidt and Spitzer [9].
The set $\mathcal{S}$ (or $\mathcal{B}$) defines the subset of the complex plane which is "filled in" by the points of σ_J as $J \to \infty$.

There are two additional sets that play major roles in the computational algorithms of the next section and they are defined in terms of the κ's. We assume without loss of generality that the $p+q$ roots of the algebraic equation (2.6) are ordered as

$$|\kappa_1(\lambda)| \le |\kappa_2(\lambda)| \le \cdots \le |\kappa_p(\lambda)| \le |\kappa_{p+1}(\lambda)| \le \cdots \le |\kappa_{p+q}(\lambda)|. \tag{3.4}$$

The crucial inequality in (3.4) is between $|\kappa_p(\lambda)|$ and $|\kappa_{p+1}(\lambda)|$, i.e.,

$$|\kappa_p(\lambda)| = |\kappa_{p+1}(\lambda)|, \quad \text{or} \tag{3.5a}$$

$$|\kappa_p(\lambda)| < |\kappa_{p+1}(\lambda)|. \tag{3.5b}$$

DEFINITION 3.2. The (boundary condition independent) set $\mathcal{C}$ is defined to be

$$\mathcal{C} = \{\lambda \mid |\kappa_p(\lambda)| = |\kappa_{p+1}(\lambda)|\}. \tag{3.6}$$

Schmidt and Spitzer [9] proved that the asymptotic Toeplitz spectrum defined by the set $\mathcal{B}$ is equivalent to the set $\mathcal{C}$, i.e.,

$$\mathcal{B} = \mathcal{C}. \tag{3.7}$$

Let

$$\mathbf{K}_{rs}(\lambda) = \begin{bmatrix} \kappa_1 & \kappa_2 & \kappa_3 & \cdots & \kappa_s \\ \kappa_1^2 & \kappa_2^2 & \kappa_3^2 & \cdots & \kappa_s^2 \\ \kappa_1^3 & \kappa_2^3 & \kappa_3^3 & \cdots & \kappa_s^3 \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \kappa_1^r & \kappa_2^r & \kappa_3^r & \cdots & \kappa_s^r \end{bmatrix}, \tag{3.8}$$

which is an $r \times s$ matrix.

DEFINITION 3.3. The (boundary condition dependent) set $\mathcal{D}$ is defined to be

$$\mathcal{D} = \{\lambda \mid |\kappa_p(\lambda)| < |\kappa_{p+1}(\lambda)|, \det[(\mathbf{B} - \lambda\mathbf{D})\mathbf{K}_{rp}] = 0 \vee \det[(\mathbf{C} - \lambda\mathbf{D})\mathbf{K}_{rq}] = 0\}, \tag{3.9}$$

where $\mathbf{B}$ and $\mathbf{C}$ are the left and right boundary matrices with p and q rows, respectively, and r columns. The matrix $\mathbf{D}$ is a $p \times r$ rectangular diagonal matrix with unit entries on the diagonal. (As examples of the left and right boundary matrices see (2.11) and (4.10).)

Kreiss [7] proved that the asymptotic spectrum for non-Dirichlet boundary conditions (the quasi-Toeplitz matrix) is equal to the asymptotic spectrum for Dirichlet boundary conditions (the Toeplitz matrix) plus a set (possibly empty) of isolated eigenvalues (set $\mathcal{D}$), i.e.,

$$\mathcal{S} = \mathcal{B} \cup \mathcal{D}. \tag{3.10}$$

With the Schmidt and Spitzer identity (3.7) we have

$$\mathcal{S} = \mathcal{C} \cup \mathcal{D}. \tag{3.11}$$

The investigation of the set $\mathcal{D}$ is an integral part of the work of Gustafsson, Kreiss, and Sundström [6]. In the next section we present algorithms for numerically determining the sets $\mathcal{C}$ and $\mathcal{D}$ and thus the asymptotic spectra for banded quasi-Toeplitz matrices.

For a pure Toeplitz matrix the asymptotic spectrum is the set $\mathcal{C}$. For the quasi-Toeplitz cousin matrix, which can have arbitrary boundary conditions, the set $\mathcal{C}$ is a subset of the asymptotic spectrum $\mathcal{S}$. We say therefore that the set $\mathcal{C}$ is the boundary condition independent part of the the asymptotic spectrum and the set $\mathcal{D}$ is the boundary condition dependent part of the asymptotic spectrum. Note that the asymptotic spectrum for a pure Toeplitz matrix is the boundary condition independent spectrum.

4. Algorithms for computing the asymptotic spectra. In this section we present algorithms for computing the asymptotic spectra of Toeplitz and quasi-Toeplitz matrices. We separate the algorithms for the two sets: (a) boundary condition independent (pure Toeplitz) set $\mathcal{C}$, and (b) boundary condition dependent set $\mathcal{D}$. The application of each algorithm involves the solution of algebraic equations whose degree is proportional to the bandwidth $p + q + 1$ of the matrix.

4.1. Boundary condition independent (Toeplitz) spectrum. The algorithm for the pure Toeplitz asymptotic spectrum (set $\mathcal{C}$) is based on the observation that separated the spectrum into two sets, i.e., we seek those eigensolutions whose κ's satisfy (3.4) with the equality (3.5a). First we seek the solutions of the algebraic equation (2.6) with two distinct but equal modulus κ's. We denote these κ's by

$$\kappa_a = \hat{\kappa} e^{i\psi_\ell}, \quad \kappa_b = \hat{\kappa} e^{-i\psi_\ell}, \quad 0 < \psi_\ell < \pi, \tag{4.1}$$

where $\hat{\kappa}$ is a complex variable. For computational convenience, the interval $[0, \pi]$ is divided into $M + 1$ subintervals of size $\Delta\psi$ where $(M + 1)\Delta\psi = \pi$ and $\psi_\ell = \ell\Delta\psi$, $\ell = 1, 2, \cdots, M$. (The choice of the integer M determines the number of computed points on the asymptotic spectrum, i.e., the resolution of the spectrum, but does not affect the accuracy of the computed points.) An equation for $\hat{\kappa}$ (independent of λ) is easily obtained. Since κ_a and κ_b must each give the same value of λ when inserted in (2.6), we can eliminate λ and obtain a polynomial in $\hat{\kappa}$ with coefficients that are functions of the Toeplitz matrix elements and ψ_ℓ. In particular,

$$\lambda(\kappa_a) = \sum_{m=-p}^{q} a_m \hat{\kappa}^m e^{im\psi_\ell}, \qquad \lambda(\kappa_b) = \sum_{m=-p}^{q} a_m \hat{\kappa}^m e^{-im\psi_\ell}, \tag{4.2}$$

and since

$$\lambda(\kappa_a) = \lambda(\kappa_b), \tag{4.3}$$

one obtains a polynomial equation of order $p+q$ for $\hat{\kappa}$:

$$\sum_{m=-p}^{q} a_m \sin(m\psi_\ell)\hat{\kappa}^m = 0. \tag{4.4}$$

The algorithm for the boundary condition independent spectrum, set $\mathcal{C}$, is the following.

ALGORITHM 4.1. For each of the M values of ψ_ℓ:

(i) Solve (4.4) for the $p+q$ roots $\hat{\kappa}$.

(ii) Check the corresponding κ's to determine if they satisfy (3.4) with (3.5a), i.e., that κ_a and κ_b from (4.1) are in fact κ_p and κ_{p+1}. This is done as follows.
For each $\hat{\kappa}$ obtained in (i)

(iia) Calculate κ_a and κ_b from (4.1) and insert κ_a (or κ_b) in (2.6) to find λ. Substitute this value for λ back into (2.6) and solve the algebraic equation to determine the remaining $(p+q-2)$ κ's.

(iib) If the $p+q$ κ's satisfy the inequality (3.4) with (3.5a), i.e., $|\kappa_a| = |\kappa_b| = |\kappa_p| = |\kappa_{p+1}|$, then λ is a point on the asymptotic Toeplitz spectrum. If the κ's do not satisfy the inequality test, discard the λ.
Return to (iia) until all $\hat{\kappa}$'s from (i) have been tested.
Replace ψ_ℓ by $\psi_{\ell+1}$ and return to step (i).

It should be emphasized that the algorithm (4.1) requires the solution of many algebraic equations but their degree is proportional to the bandwidth of the matrix and not the order of the matrix. Note that the boundary conditions do not enter the calculation. An implementation of the algorithm in MATLAB is given in the Appendix. The algorithm will become more transparent in the examples of §6.

4.2. Boundary condition dependent spectrum. The algorithm for the boundary condition dependent asymptotic spectrum (set $\mathcal{D}$) closely parallels the eigenvalue analysis of Kreiss [7] and Gustafsson, Kreiss, and Sundström [6]. There are however some differences since their analysis for hyperbolic initial-boundary-value problems makes two assumptions: (I) Cauchy stability (an assumption about the spectrum of the Toeplitz matrix's circulant cousin), and (II) the quarter-plane eigensolutions are *unstable* (i.e., the eigenvalues are in a particular part of the complex plane). Since we are interested in the entire spectrum for any quasi-Toeplitz matrix, we do not impose conditions (I) and (II). The relaxation of these two conditions leads to a slightly more complicated algorithm.

The algorithm for the boundary condition dependent part of the spectrum is straightforward. We seek p (or q) values of κ that satisfy (2.6) and satisfy the left p (or right q) boundary difference equations. The remaining q (or p) κ's must satisfy

(3.4) with inequality (3.5b). The algorithm for the eigenvalues associated with the *left* boundary is as follows.

ALGORITHM 4.2.

(i) Assume a solution of the form

$$\phi_j = \sum_{m=1}^{p} \beta_m \kappa_m^j. \tag{4.5}$$

Substitute ϕ_j from (4.5) into the p left boundary difference equations (2.11) to obtain

$$(\mathbf{B} - \lambda\mathbf{D})\mathbf{K}_{rp}\boldsymbol{\beta} = 0, \tag{4.6}$$

where $\mathbf{K}_{rp}$ is defined by (3.8) and

$$\boldsymbol{\beta}^T = [\beta_1, \beta_2, \cdots, \beta_p]. \tag{4.7}$$

Equation (4.6) is a system of p equations for the $2p+1$ unknowns: p κ_m's, p β_m's, and λ. This system of p equations is linear and homogeneous in the β's and therefore a nontrivial solution exists if and only if

$$\det[(\mathbf{B} - \lambda\mathbf{D})\mathbf{K}_{rp}] = 0. \tag{4.8}$$

The determinant condition leads to a single equation for the p κ's and λ. An additional p equations are required and they are derived from the algebraic equation (2.6) since each of the p κ's must give the same value of λ. This system of polynomial equations can be reduced to a single polynomial equation for a single variable and we choose κ_p.

(ii) The roots of the polynomial equation from (i) contain all *candidate* κ_p's for the (left) boundary condition dependent spectrum. For each κ_p the corresponding κ_m's must be checked. This is accomplished by calculating the corresponding λ ($\kappa = \kappa_p$) from (2.6) and then computing the $p+q$ roots of (2.6) for the given λ. Finally, we compare those roots with the p κ_m's which satisfy the boundary difference equations (2.11). If (3.4) and (3.5b) are both satisfied, the λ is an eigenvalue in the boundary condition dependent part of the spectrum.

The algorithm for the eigenvalues associated with the right boundary is similar. If the algorithm has been implemented for the left boundary problem, it can be used for the right boundary problem by pre- and post-multiplying $\mathbf{A}$ by the permutation matrix

$$\mathbf{P} = \begin{bmatrix} & & & & 1 \\ & O & & 1 & \\ & & \cdot & & \\ & & \cdot & & \\ & \cdot & & & \\ & 1 & & O & \\ 1 & & & & \end{bmatrix}, \tag{4.9}$$

i.e., $\mathbf{PAP}$. Note that $\mathbf{P}^{-1} = \mathbf{P}$. The matrix $\mathbf{PAP}$ is most easily obtained by interchanging the columns of $\mathbf{A}$ "left to right" and then interchanging the rows "up and down." The boundary difference equations associated with the left boundary for **PAP** are

$$\mathbf{C}\tilde{\phi} = \lambda \mathbf{D}\tilde{\phi}. \tag{4.10}$$

As an example, the matrices **C** and **D** associated with **PAP** where **A** is (1.4) are

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{4.11}$$

This explains the somewhat unconventional indexing for the c elements of (1.4). (Note that a pure Toeplitz matrix **A** is persymmetric, i.e., it is symmetric about its northeast-southwest diagonal [3]. Consequently $\mathbf{A}^T = \mathbf{PAP}$. However if **A** is *quasi*-Toeplitz then **PAP** is *not* the transpose of **A**.)

The details of the algorithm are illustrated by the examples of §6. Note that there may be no eigenvalues in the boundary condition dependent part of the spectrum even if non-Dirichlet boundary conditions are specified, i.e., the set $\mathcal{D}$ may be empty.

The algorithm for the boundary condition dependent spectrum is straightforward, but finding a polynomial equation in a single variable can be algebraically involved and the degree of the polynomial can be large. The system of equations for the κ's in (i) are symmetric polynomials; therefore, the algebraic complexity can be somewhat reduced by the use of the elementary symmetric functions. We have implemented Algorithm 4.2 in MACSYMA, and it performs satisfactorily on an IRIS workstation for the matrix **B** with two rows and a quasi-Toeplitz matrix **A** with bandwidths up to five.

4.3. Alternative boundary condition dependent algorithm. The algebraic complexity of the boundary condition dependent algorithm of §4.2 led us to consider alternative methods. In the GKS (Gustafsson, Kreiss and Sundström [6]) theory a GKS (i.e., *unstable*) eigenvalue is a member of the boundary condition dependent eigenvalue set $\mathcal{D}$ of this paper. Kreiss [7] recognized that it is often very difficult to find GKS eigenvalues by analytical methods. He proposed finding GKS eigenvalues (if they exist) by a numerical eigenvalue computation of the finite-domain problem where the boundary condition (or conditions) to be checked are on the left boundary and Dirichlet boundary conditions are imposed on the right boundary. Kreiss showed that GKS eigenvalues converge exponentially fast with increasing J. However, we found difficulties in numerically delineating the GKS eigenvalues (or other boundary condition dependent eigenvalues) for *nonnormal* matrices. The numerical difficulties are often alleviated if the matrix is rescaled as described in §7.

We have found the following algorithm to be useful although not always completely reliable since it will sometimes "miss" an eigenvalue. The reliability is improved if the algorithm is repeated for several matrix orders.

ALGORITHM 4.3.
(i) Place the boundary conditions of interest on the left boundary and Dirichlet conditions on the right boundary.
(ii) Apply the similarity transformation described in §7 and use a conventional eigenvalue algorithm to find the spectrum for large J.
(iii) Test each eigenvalue found in (ii) using part (ii) of Algorithm 4.2.

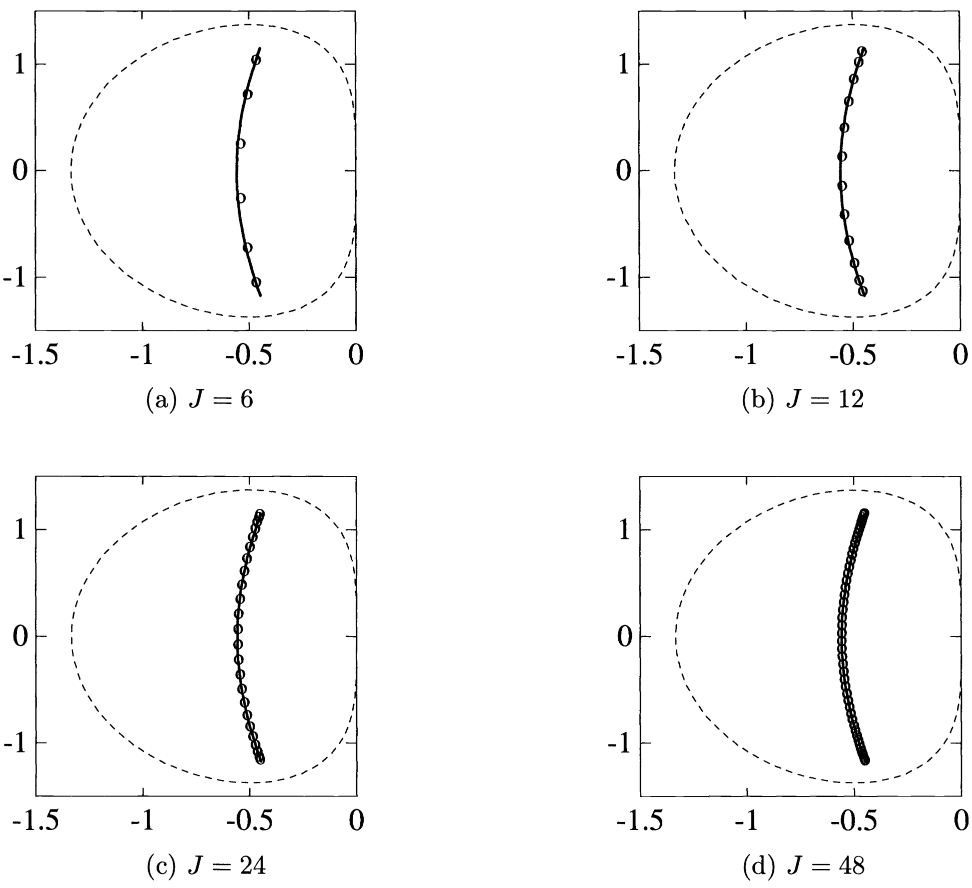
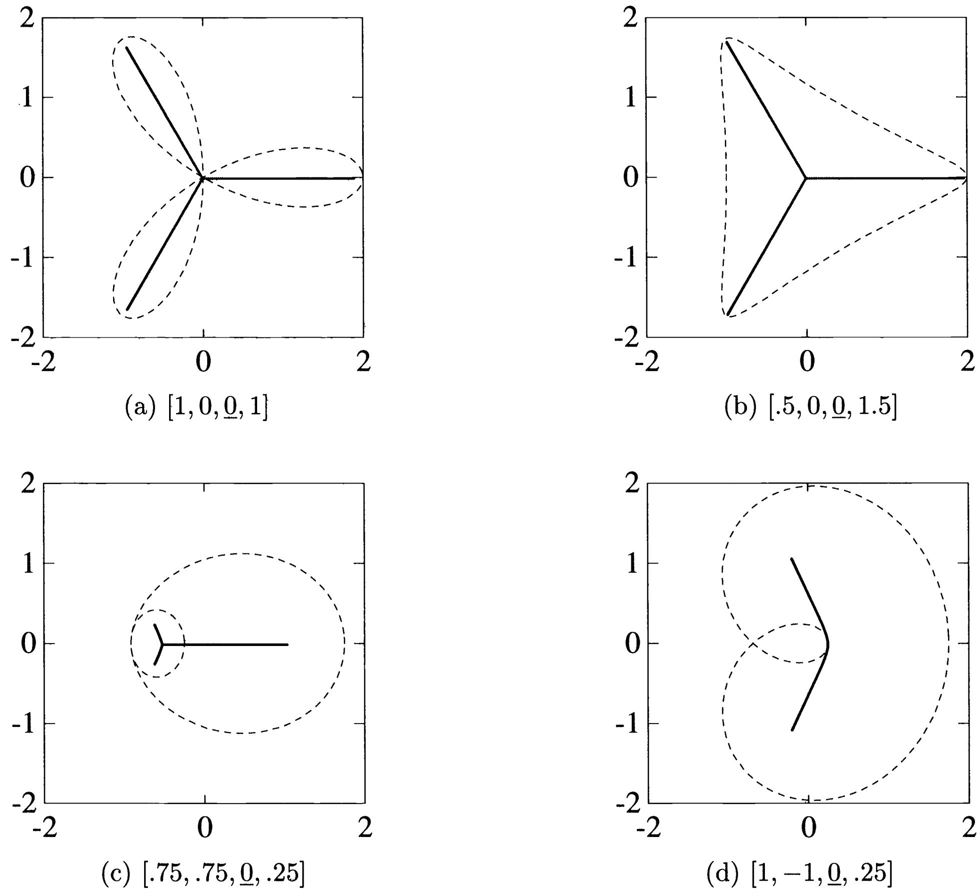


FIG. 5.1. *Toeplitz matrix* $[-1/3, \underline{-1/2}, 1, -1/6]$ *spectra "fill in" for increasing matrix order* J. *The symbol* o *denotes an eigenvalue. Asymptotic spectra* (*Algorithm 4.1*) *are denoted by solid curves and asymptotic circulant spectra* (2.17) *by dashed curves.*

5. Numerical application of algorithms. Plots of the asymptotic spectra of Toeplitz matrices present some interesting and sometimes surprising geometrical shapes. In this section we present some spectra that were obtained with the algorithms described in §4. The matrix coefficients were selected more or less at random to give a flavor of the geometric patterns that may occur.

In each figure we show the spectra plotted in the complex plane. Open circles represent eigenvalues computed using MATLAB $eig(A_J)$ for the matrix order (J) specified in the figure legend. The "solid" curve (which is formed by closely spaced dots) is the asymptotic spectrum for the Toeplitz matrix and was computed using Algorithm 4.1. The star symbol ($*$) is used to represent the boundary condition dependent part of the asymptotic spectrum computed using Algorithm 4.2 or 4.3. The dashed curve represents the asymptotic spectrum for the circulant cousin of the pure Toeplitz matrix.

5.1. Toeplitz spectra. First we summarize the known properties of the asymp-



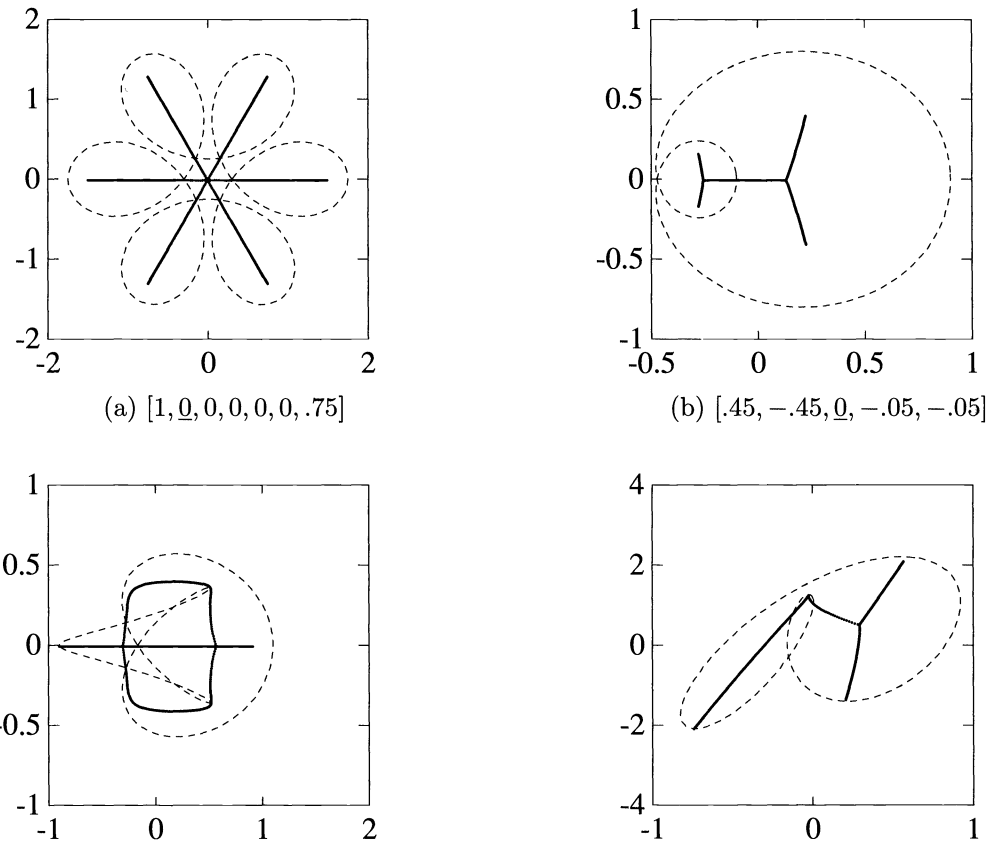
(a) $[1, 0, \underline{0}, 1]$ (b) $[.5, 0, \underline{0}, 1.5]$

(c) $[.75, .75, \underline{0}, .25]$ (d) $[1, -1, \underline{0}, .25]$

FIG. 5.2. *Asymptotic spectra for pure Toeplitz matrices (Algorithm 4.1) indicated by solid curves and the asymptotic spectra for the circulant cousin Toeplitz matrices indicated by dashed curves.*

totic spectra of pure Toeplitz matrices and show some examples that illustrate these properties. Definition 3.1 involves a family of matrices $\mathbf{A}_J$ where the dimension J tends to infinity. As J increases a subset of the complex plane is "filled in" by the spectrum σ_J and as $J \to \infty$ one obtains the asymptotic spectrum [9]. This is illustrated in Fig. 5.1 for a quadridiagonal pure Toeplitz matrix defined by the sequence (2.16). Note that the spectrum σ_J is close to the asymptotic spectrum even for small values of J.

The asymptotic spectrum of a banded pure Toeplitz matrix is compact and locally it consists of analytical arcs [9]. Furthermore, it possesses no isolated points [9]. This property will be of interest in considering the asymptotic spectra of quasi-Toeplitz matrices. Ullman [14] has shown that the asymptotic spectrum is connected. Asymptotic spectra illustrating these properties are plotted in Figs. 5.2 and 5.3. The defining sequence (1.1) is shown under each figure. Algorithm 4.1 is not restricted to Toeplitz matrices with *real* coefficients and this is demonstrated in Fig. 5.3(d). As the bandwidth of the matrix increases, the asymptotic spectrum becomes increasingly



(a) $[1, \underline{0}, 0, 0, 0, 0, .75]$

(b) $[.45, -.45, \underline{0}, -.05, -.05]$

(c) $[-.2, -.2, -.2, \underline{.1}, -.2, .2, -.4]$

(d) $[.7i, .3, \underline{0}, .5i, .3 + i]$

FIG. 5.3. *Asymptotic spectra for Toeplitz matrices (Algorithm 4.1) indicated by solid curves and their circulant cousin Toeplitz matrices indicated by dashed curves.*

complex.

According to Schmidt and Spitzer [9] it is not easy to give a simple geometric description of the asymptotic Toeplitz spectrum for nonnormal matrices. An exception is the special case where the defining sequence is $[a_{-p}, 0, \cdots, 0, a_q]$, i.e., all members of the sequence are zero except the two "outriders." In this case the spectrum is a star-shaped figure. Examples of star-shaped spectra for quadridiagonal matrices are shown in Figs. 5.2(a) and 5.2(b) and for a septadiagonal matrix in Fig. 5.3(a). The spectrum for the tridiagonal case is known analytically and consists of a straight line segment (connecting the foci of the ellipse which is the spectrum of its circulant cousin) [16].

It is a general result [9] that the asymptotic Toeplitz spectrum is "enclosed" by the spectrum of its circulant cousin. This is illustrated in Figs. 5.2 and 5.3. However, for finite J the spectrum σ_J is not necessarily enclosed by the asymptotic spectrum of the circulant cousin of $\mathbf{A}$. This is illustrated in Fig. 5.4.

5.2. Quasi-Toeplitz spectra. The properties of the asymptotic spectrum of a

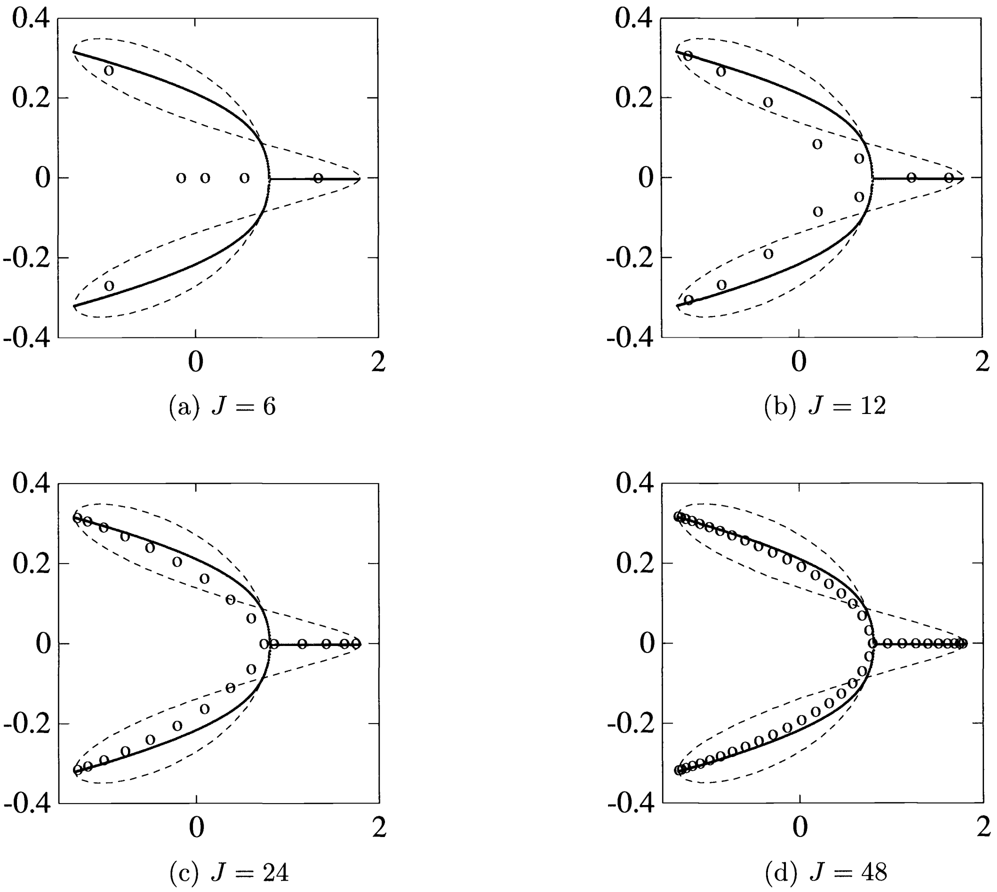


FIG. 5.4. *Spectra for finite matrix order J (o symbol) and the asymptotic circulant spectrum (dashed curves). Asymptotic Toeplitz spectra indicated by solid curves. The Toeplitz matrix is* $[.7, .1, \underline{0}, .4, .6]$.

quasi-Toeplitz matrix are illustrated in Fig. 5.5 for the matrix (1.7). The asymptotic spectrum consists of the asymptotic pure Toeplitz spectrum, set $\mathcal{C}$, plus (possible) isolated eigenvalues corresponding to boundary condition dependent eigenvalues, set $\mathcal{D}$. The isolated eigenvalues (see, e.g., Fig. 5.5(b)) must be boundary condition dependent eigenvalues associated with set $\mathcal{D}$ because the pure Toeplitz spectrum can have no isolated points [9]. Note also that, in contrast to the pure Toeplitz spectrum, the quasi-Toeplitz spectrum may have (isolated) eigenvalues outside the circulant spectrum, e.g., Figs. 5.5(b), 5.5(c), and 5.5(d). For $\alpha = \beta = 0$ the matrix (1.7) has no boundary condition dependent eigenvalues and the asymptotic quasi-Toeplitz spectrum is the pure Toeplitz spectrum shown in Fig. 5.5(a). In this example set $\mathcal{D}$ is empty. A fundamental property of the eigenvalues associated with set $\mathcal{D}$ is that the "left" and "right" boundary dependent eigenvalues are uncoupled. This is illustrated in Figs. 5.5(b), 5.5(c), and 5.5(d). For $\beta = -.8$ there are three isolated eigenvalues to the right of the pure Toeplitz spectrum. These eigenvalues are completely independent of the parameter α of the left boundary condition (see matrix (1.7)). This fact

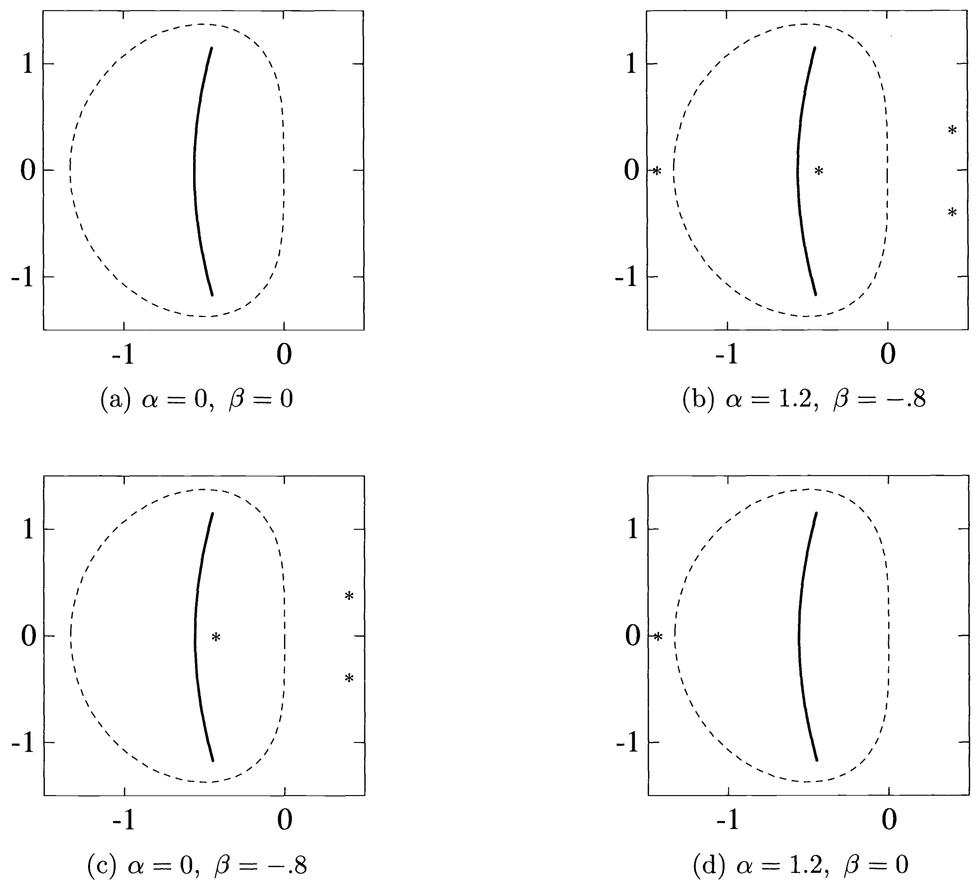


FIG. 5.5. *Asymptotic spectra for the quasi-Toeplitz matrix* (1.7). *Boundary condition independent spectra computed with Algorithm 4.1* (*solid curve*), *boundary condition dependent spectra* ($*$ *symbol*) *computed with Algorithm 4.2, and asymptotic circulant spectra* (*dashed curves*).

is illustrated by comparing Figs. 5.5(b) and 5.5(c) where β is fixed and α is changed. For $\alpha = 1.2$ there is one isolated eigenvalue to the left of the circulant spectrum. This eigenvalue is independent of the parameter β of the right boundary condition as is indicated by comparing Figs. 5.5(b) and 5.5(d).

6. Analytical application of algorithms. If the bandwidth of the matrix is sufficiently small (≈ 3) the algorithms of §4 can be used as analytical methods. In this section we elucidate the algorithms of the previous section by considering small bandwidth examples for both Toeplitz and quasi-Toeplitz matrices.

6.1. Toeplitz tridiagonal matrix. A tridiagonal Toeplitz matrix is defined by the sequence $[a_{-1}, a_0, a_1]$ where $p = q = 1$ (see (1.1)). For this case the spectrum for arbitrary order J can be determined analytically. The solution is "well known" although the eigenvalue formula does not appear in many references. A derivation is given by Smith [10, p. 113]. The algebraic equation (2.6) becomes

$$\lambda = a_{-1}\kappa^{-1} + a_0 + a_1\kappa. \tag{6.1}$$

By rewriting (6.1), one obtains

$$a_1\kappa^2 + (a_0 - \lambda)\kappa + a_{-1} = 0. \tag{6.2}$$

We denote the roots of this quadratic by κ_1, κ_2. The product of the roots is

$$\kappa_1\kappa_2 = \frac{a_{-1}}{a_1}. \tag{6.3}$$

We follow the algorithm for the boundary condition independent part of the spectrum from §4.1. Equation (4.4) for $\hat{\kappa}$ reduces to

$$a_1\hat{\kappa}^2 - a_{-1} = 0. \tag{6.4}$$

In step (i) we solve for the roots of (6.4), i.e.,

$$\hat{\kappa} = \pm\sqrt{a_{-1}/a_1}. \tag{6.5}$$

In step (iia) we find

$$\kappa_a = \sqrt{a_{-1}/a_1}\,e^{i\psi_\ell}, \tag{6.6a}$$

$$\kappa_b = \sqrt{a_{-1}/a_1}\,e^{-i\psi_\ell}, \tag{6.6b}$$

where we have chosen the positive square root. (The negative square root gives the same spectrum.) Since there are only two roots of (6.1), i.e., κ_1 and κ_2, they must be κ_a and κ_b. They obviously have equal modulus, therefore equality (3.5a) is satisfied and the corresponding λ is part of the Toeplitz spectrum. (In this simple example step (ii) of the Algorithm 4.1 is rather trivial.) Substitution of κ_a given by (6.6a) into (6.1) yields

$$\lambda_\ell = a_0 + 2\sqrt{a_1 a_{-1}}\cos\psi_\ell, \quad 0 < \psi_\ell < \pi, \tag{6.7}$$

which is the asymptotic spectrum for a tridiagonal Toeplitz matrix.

Although not part of the asymptotic analysis we note that for the tridiagonal case, the roots κ_1 and κ_2 have equal modulus regardless of the order J of the matrix. In fact, the exact roots are given by

$$\kappa_1 = \hat{\kappa}e^{i\psi_\ell}, \qquad \kappa_2 = \hat{\kappa}e^{-i\psi_\ell}, \quad \text{where} \quad \psi_\ell = \ell\pi/(J+1), \quad \ell = 1, 2, \cdots, J, \tag{6.8}$$

and $\hat{\kappa}$ is given by (6.5) (see, e.g. [16]). Consequently, the exact spectrum for a $J \times J$ matrix is (6.7) where ψ_ℓ is defined in (6.8). It should be emphasized that if the bandwidth of a Toeplitz matrix is larger than three, then Algorithm 4.1 is, in general, exact only in the asymptotic limit $J \to \infty$.

6.2. Quasi-Toeplitz tridiagonal matrix. The asymptotic spectrum for a quasi-Toeplitz tridiagonal matrix is given by (6.7) plus any boundary condition dependent eigenvalues, i.e., set $\mathcal{D}$. We first show that a pure Toeplitz tridiagonal matrix does not have any boundary condition dependent eigenvalues, i.e., the homogeneous (Dirichlet) boundary conditions (2.4) introduce no boundary condition dependent eigenvalues. For a tridiagonal matrix the homogeneous conditions (2.4) are simply

$$\phi_0 = 0, \tag{6.9a}$$

$$\phi_{J+1} = 0. \tag{6.9b}$$

By substituting (4.5), i.e.,

$$\phi_j = \beta_1 \kappa_1^j \tag{6.10}$$

into (6.9a), one concludes that either $\beta_1 = 0$ or $\kappa_1 = 0$. But the product of the roots (6.3) is nonzero and $\beta_1 = 0$. Hence there is no nontrivial eigenfunction and no eigenvalue is introduced by the (left) boundary condition (6.9a). By substituting (6.10) into (6.9b), one finds the similar result that no eigenvalue is introduced by the (right) boundary condition (6.9b).

For the general Toeplitz matrix there are p homogeneous Dirichlet boundary conditions at the left boundary and q homogeneous Dirichlet boundary conditions at the right boundary, i.e., (2.4a) and (2.4b). Substitution of (4.5) into the left boundary conditions leads to a homogeneous system $\mathbf{V}\boldsymbol{\beta} = 0$ where $\mathbf{V}$ is a $p \times p$ Vandermonde matrix and $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_p]^T$. The determinant of $\mathbf{V}$ equals zero if and only if two κ's are equal which violates the assumption of distinct κ's. The assumption of nondistinct κ's also leads to the trivial case $\boldsymbol{\beta} = 0$. Therefore Dirichlet boundary conditions introduce no boundary condition dependent eigenvalues, i.e., the set $\mathcal{D}$ is empty.

Consider now boundary conditions that introduce nontrivial solutions. Suppose that when (6.10) is introduced into the left boundary condition we obtain

$$\kappa_1 = \kappa_\sigma, \tag{6.11}$$

where κ_σ is some nonzero complex constant. Does this introduce an eigenvalue? It does if inequality (3.5b) is satisfied, i.e., $|\kappa_1| < |\kappa_2|$. From (6.3) one obtains

$$|\kappa_2| = |a_{-1}/a_1|/|\kappa_\sigma|. \tag{6.12}$$

Inequality (3.5b) is satisfied if

$$|\kappa_1| = |\kappa_\sigma| < \sqrt{|a_{-1}/a_1|} \tag{6.13}$$

and the corresponding eigenvalue λ is obtained by substituting κ_σ into (6.1).

We give an example of a quasi-Toeplitz matrix where there is a boundary condition dependent eigenvalue. Consider the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & -2 & 2 & & & \\ -1 & 0 & 1 & & O & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & O & & -1 & 0 & 1 \\ & & & & -1 & 0 \end{bmatrix}. \tag{6.14}$$

Here

(6.15) $$a_{-1} = -1, \quad a_0 = 0, \quad a_1 = 1, \quad b_{11} = 0, \quad b_{12} = -2, \quad b_{13} = 2.$$

The eigenvalue problem is (2.3) with $p = q = 1$, and the boundary difference equation (2.11) is

(6.16) $$-2\phi_2 + 2\phi_3 = \lambda\phi_1.$$

Substitution of (6.10) into (6.16) yields

(6.17) $$-2\kappa_1 + 2\kappa_1^2 = \lambda.$$

The second equation relating κ_1 and λ is (6.2) with $\kappa = \kappa_1$ and coefficients (6.15)

(6.18) $$\kappa_1^2 - \lambda\kappa_1 - 1 = 0.$$

Elimination of λ from (6.17) and (6.18) yields

(6.19) $$(\kappa_1 - 1)^2(2\kappa_1 + 1) = 0.$$

The roots of (6.19) are

(6.20) $$\kappa_1 = 1, 1, -1/2.$$

In the tridiagonal case, step (ii) of Algorithm 4.2 is simplified to checking inequality (6.13). The inequality (6.13), with (6.15),

$$|\kappa_1| = |\kappa_\sigma| < 1$$

is satisfied for $\kappa_1 = -1/2$. Substitution of $\kappa_1 = -1/2$ into (6.17) gives $\lambda = 3/2$.

The asymptotic spectrum of the quasi-Toeplitz matrix (6.14) consists of two parts. The boundary condition independent (pure Toeplitz) spectrum, set $\mathcal{C}$, is (6.7)

(6.21) $$\lambda = i2\cos\psi, \quad 0 < \psi < \pi,$$

and the boundary condition dependent part of the spectrum, set $\mathcal{D}$, is $\lambda = 3/2$. Consequently, the asymptotic spectrum is a line segment on the imaginary axis between $\pm 2i$ and the single eigenvalue $\lambda = 3/2$.

6.3. Toeplitz quadridiagonal matrix. A more interesting example is the quadridiagonal matrix (1.3) where $p = 1$ and $q = 2$. The algebraic equation (2.6) is

(6.22) $$\lambda = a_{-1}\kappa^{-1} + a_0 + a_1\kappa + a_2\kappa^2.$$

For the boundary condition independent spectrum (set $\mathcal{C}$) , the three roots of the cubic (6.22) must satisfy (3.4) and (3.5a)

(6.23) $$|\kappa_1| = |\kappa_2| \le |\kappa_3|.$$

The product of the roots of (6.22) is $-a_{-1}/a_2$ and therefore

(6.24) $$|\kappa_1||\kappa_2||\kappa_3| = |a_{-1}/a_2|.$$

The polynomial (4.4) for $\hat{\kappa}$ becomes

$$a_{-1}\sin(-\psi_\ell)\hat{\kappa}^{-1} + a_1\sin(\psi_\ell)\hat{\kappa} + a_2\sin(2\psi_\ell)\hat{\kappa}^2 = 0$$

or

$$2a_2(\cos\psi_\ell)\hat{\kappa}^3 + a_1\hat{\kappa}^2 - a_{-1} = 0. \tag{6.25}$$

In (i) of Algorithm 4.1 we solve (6.25) numerically for specified coefficients a_ℓ.

In (iia) of Algorithm 4.1 we proceed as follows. For each root $\hat{\kappa}$ we substitute $\kappa_a = \hat{\kappa}e^{i\psi_\ell}$ into (6.22) and calculate the corresponding λ. Then with λ given, we solve the cubic (6.22) for the roots $\kappa_1, \kappa_2, \kappa_3$. Of course, two of the roots are already known, i.e., $\kappa_a = \hat{\kappa}e^{i\psi_\ell}$ and $\kappa_b = \hat{\kappa}e^{-i\psi_\ell}$. In (iib) we check to see if the three roots satisfy inequality (6.23)

$$|\kappa_a| = |\kappa_b| = |\kappa_1| = |\kappa_2| \le |\kappa_3|. \tag{6.26}$$

If this inequality is satisfied, then the corresponding λ is a point of the boundary condition independent spectrum. If the κ's fail to satisfy inequality (6.26), we discard the λ. We repeat (i) and (ii) for all of the ψ_ℓ's where $\ell = 1, 2, \cdots, M$.

For the particular example under consideration, the test (6.26) can be simplified as follows. Recall that the roots of equal modulus κ_1 and κ_2 are denoted by (4.1) and rewrite (6.23) as

$$|\hat{\kappa}| = |\hat{\kappa}| \le |\kappa_3| \tag{6.27}$$

and (6.24) as

$$|\hat{\kappa}||\hat{\kappa}||\kappa_3| = |a_{-1}/a_2|. \tag{6.28}$$

Inequality (6.27) is satisfied if

$$|\hat{\kappa}| \le |a_{-1}/a_2|^{1/3}. \tag{6.29}$$

Hence if $|\hat{\kappa}|$ satisfies (6.29), then the corresponding λ computed by inserting $\kappa_a = \hat{\kappa}e^{i\psi_\ell}$ into (6.22) is a point of the boundary condition independent spectrum.

The asymptotic spectrum for the Toeplitz matrix (1.3) with coefficients (2.16) was computed with Algorithm 4.1 (see also the Appendix) and is the solid curve in each of the figures of Fig. 5.1. The asymptotic spectrum for the circulant cousin (1.6) is the dashed curve in each of the figures of Fig. 5.1.

6.4. Quasi-Toeplitz quadridiagonal matrix. An example of a quasi-Toeplitz quadridiagonal matrix is (1.4) where $p = 1$ and $q = 2$. The eigenvalue problem is equivalent to (2.3), i.e.,

$$\lambda\phi_j = a_{-1}\phi_{j-1} + a_0\phi_j + a_1\phi_{j+1} + a_2\phi_{j+2}, \qquad j = 2, 3, \cdots, J \tag{6.30}$$

with left boundary difference equation (2.11)

$$b_{11}\phi_1 + b_{12}\phi_2 + b_{13}\phi_3 + b_{14}\phi_4 = \lambda\phi_1. \tag{6.31}$$

The spectrum for the matrix (1.4) is the spectrum for the pure Toeplitz matrix (plotted in Fig. 5.5(b) for coefficients (2.16)) plus any boundary condition dependent eigenvalues. For the boundary condition dependent spectrum the roots of the cubic (6.22) must satisfy

$$|\kappa_1| < |\kappa_2| \leq |\kappa_3| \tag{6.32}$$

(see (3.4) and (3.5b)).

In the analysis of the boundary condition dependent part of the spectrum, the left and right boundary conditions are uncoupled and consequently we consider each boundary separately.

6.4.1. Left boundary dependent spectrum. The left boundary difference equation is given by (6.31). We follow the algorithm for the boundary condition dependent part of the spectrum given in §4.2. Recall that for this example $p = 1$ and in the first step (i), we look for a solution of the form (4.5)

$$\phi_j = \beta_1 \kappa_1^j. \tag{6.33}$$

Insertion of this equation into the boundary difference equation (6.31) yields

$$b_{11} + b_{12}\kappa_1 + b_{13}\kappa_1^2 + b_{14}\kappa_1^3 = \lambda. \tag{6.34}$$

As a particular example we consider the special case for the matrix (1.4) where the coefficients a_ℓ are given by (2.16) and the coefficients b_{1j} are given by

$$b_{11} = -\alpha - 3/2, \quad b_{12} = 3\alpha + 2, \quad b_{13} = -3\alpha - 1/2, \quad b_{14} = \alpha. \tag{6.35}$$

The algebraic equation (6.22) with $\kappa = \kappa_1$ becomes

$$6\lambda = -\frac{2}{\kappa_1} - 3 + 6\kappa_1 - \kappa_1^2 = -(\kappa_1 - 1)(\kappa_1^2 - 5\kappa_1 - 2)/\kappa_1, \tag{6.36}$$

and the boundary difference equation (6.34) with coefficients (6.35) becomes

$$-(\alpha + 3/2) + (3\alpha + 2)\kappa_1 - (3\alpha + 1/2)\kappa_1^2 + \alpha\kappa_1^3 = \lambda. \tag{6.37}$$

If we eliminate λ from (6.36) and (6.37), the polynomial for κ_1 is

$$(\kappa_1 - 1)^3(3\alpha\kappa_1 - 1) = 0. \tag{6.38}$$

Note that if we formulate the problem in terms of the extrapolation boundary conditions (2.9), then (6.38) follows by inserting (6.33) into (2.9a).

Example 6.1. As our first example we let $\alpha = 0$ and (6.38) becomes

$$(\kappa_1 - 1)^3 = 0. \tag{6.39}$$

The repeated roots of (6.39) are

$$\kappa_1 = 1, \tag{6.40}$$

and from (6.36) the corresponding eigenvalue is $\lambda = 0$. Recall that the roots of the cubic equation (6.36) are denoted by κ_1, κ_2, and κ_2. By inserting $\lambda = 0$ into (6.36), one finds that the roots are

$$\kappa_1 = 1, \tag{6.41a}$$

$$\kappa_2 = (5 - \sqrt{33})/2 \approx -0.372, \tag{6.41b}$$

$$\kappa_3 = (5 + \sqrt{33})/2 \approx 5.372. \tag{6.41c}$$

It is obvious that inequality (6.32) is not satisfied. Therefore, if $\alpha = 0$ the left boundary condition does not introduce any boundary condition dependent eigenvalues.

Example 6.2. Although the boundary condition (6.34) with (6.35) and $\alpha = 0$ does not introduce any boundary condition dependent eigenvalues, they are introduced for a range of nonzero values of the parameter α. The roots of (6.38) are

$$\kappa_1 = 1,\ 1,\ 1,\ 1/(3\alpha) \qquad (\alpha \neq 0). \tag{6.42}$$

Are there any values of α for which $\lambda = 0$ is a boundary condition dependent eigenvalue? For $\lambda = 0$, the roots of (6.36) are

$$\kappa = 1, \quad \kappa = (5 - \sqrt{33})/2 \approx -0.372, \quad \kappa = (5 + \sqrt{33})/2 \approx 5.372. \tag{6.43}$$

It is obvious that inequality (6.32) is not satisfied if $\kappa_1 = 1$ which excludes the repeated roots in (6.42). Inequality (6.32) is satisfied if

$$\kappa_1 = (5 - \sqrt{33})/2 \approx -0.372, \tag{6.44a}$$

$$\kappa_2 = 1, \tag{6.44b}$$

$$\kappa_3 = (5 + \sqrt{33})/2 \approx 5.372. \tag{6.44c}$$

In order to have a boundary condition dependent eigenvalue $\lambda = 0$ we must choose α in (6.42), i.e.,

$$\kappa_1 = 1/(3\alpha) \tag{6.45}$$

so that κ_1 is (6.44a). By equating (6.45) and (6.44a), one obtains

$$\alpha = -(5 + \sqrt{33})/12 \approx -0.895. \tag{6.46}$$

Consequently, for this value of α the boundary condition (6.34) with (6.35) will introduce the boundary condition dependent eigenvalue $\lambda = 0$.

If one carries out a GKS normal mode analysis for the semidiscrete approximation with (2.16) and the numerical boundary condition (6.34) with (6.35), one finds that the semidiscrete approximation is unstable for $\alpha < \hat{\alpha}$ where $\hat{\alpha}$ is defined by (6.46).

An explicit formula for λ in terms of the parameter α is found by substituting (6.45) into (6.37)

$$\lambda = (3\alpha - 1)(-18\alpha^2 - 15\alpha + 1)/(54\alpha^2). \tag{6.47}$$

If one chooses a particular value of α, the corresponding value of λ is a boundary condition dependent eigenvalue if and only if the corresponding κ's satisfy inequality (6.32).

Example 6.3. Is it possible to choose α so that there is an eigenvalue

$$\lambda = -4/3 \tag{6.48}$$

which falls on the oval shaped circulant spectrum where the *oval* crosses the real axis (see Fig. 5.5a)? This value of λ is an eigenvalue of the circulant matrix (1.6) with (2.16) as one can easily verify from the formula (2.17) for $\theta = \pi$. (The corresponding κ for the circulant case is $\kappa = -1$.) By substituting $\lambda = -4/3$ into the cubic equation (6.36) one obtains the roots

$$\kappa = -1, \tag{6.49a}$$

$$\kappa = (7 - \sqrt{41})/2 \approx .298, \tag{6.49b}$$

$$\kappa = (7 + \sqrt{41})/2 \approx 6.70. \tag{6.49c}$$

Inequality (6.32) is satisfied if and only if

$$\kappa_1 = (7 - \sqrt{41})/2 \approx .298, \tag{6.50a}$$

$$\kappa_2 = -1, \tag{6.50b}$$

$$\kappa_3 = (7 + \sqrt{41})/2 \approx 6.70. \tag{6.50c}$$

If α in (6.45) is chosen so that κ_1 is (6.50a), one obtains

$$\alpha = 1/(3\kappa_1) = (7 + \sqrt{41})/12 \approx 1.117. \tag{6.51}$$

Hence $\lambda = -4/3$ is a point of the boundary condition dependent spectrum for the value of α given by (6.51).

6.4.2. Right boundary dependent spectrum. In the analysis for the boundary condition dependent part of the spectrum, the left and right boundary conditions are uncoupled. It is convenient to have a single algorithm for both the left and right boundary condition problems. This can be accomplished by *switching* the matrix so that the right and left boundary conditions are interchanged. For example, if the matrix (1.4) is switched as described in §4.2 one obtains the the matrix

$$\widehat{\mathbf{A}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & & & & \\ c_{21} & c_{22} & c_{23} & c_{24} & & & O & \\ \hat{a}_{-2} & \hat{a}_{-1} & \hat{a}_0 & \hat{a}_1 & & & & \\ \cdot & \cdot & \cdot & \cdot & & & & \\ & \cdot & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & \cdot & & \\ & O & & \hat{a}_{-2} & \hat{a}_{-1} & \hat{a}_0 & \hat{a}_1 & \\ & & & & \hat{a}_{-2} & \hat{a}_{-1} & \hat{a}_0 & \hat{a}_1 \\ & & & & b_{14} & b_{13} & b_{12} & b_{11} \end{bmatrix}, \tag{6.52}$$

where $\hat{a}_j = a_{-j}$. For this matrix $p = 2$ and $q = 1$ and the left boundary difference equations are given by

$$c_{11}\phi_1 + c_{12}\phi_2 + c_{13}\phi_3 + c_{14}\phi_4 = \lambda\phi_1, \tag{6.53a}$$

$$c_{21}\phi_1 + c_{22}\phi_2 + c_{23}\phi_3 + c_{24}\phi_4 = \lambda\phi_2. \tag{6.53b}$$

For the matrix (6.52), the algebraic equation (2.6) is

$$\lambda = \hat{a}_{-2}\kappa^{-2} + \hat{a}_{-1}\kappa^{-1} + \hat{a}_0 + \hat{a}_1\kappa. \tag{6.54}$$

There are two boundary conditions (6.53) and following the algorithm for the boundary condition dependent spectrum we seek a solution of the form (4.5), i.e.,

$$\phi_j = \beta_1\kappa_1^j + \beta_2\kappa_2^j. \tag{6.55}$$

From (6.53a) and (6.53b) one obtains

$$\begin{aligned} c_{11}(\beta_1\kappa_1 + \beta_2\kappa_2) + c_{12}(\beta_1\kappa_1^2 + \beta_2\kappa_2^2) + c_{13}(\beta_1\kappa_1^3 + \beta_2\kappa_2^3) \\ + c_{14}(\beta_1\kappa_1^4 + \beta_2\kappa_2^4) = \lambda(\beta_1\kappa_1 + \beta_2\kappa_2), \end{aligned} \tag{6.56a}$$

$$\begin{aligned} c_{21}(\beta_1\kappa_1 + \beta_2\kappa_2) + c_{22}(\beta_1\kappa_1^2 + \beta_2\kappa_2^2) + c_{23}(\beta_1\kappa_1^3 + \beta_2\kappa_2^3) \\ + c_{24}(\beta_1\kappa_1^4 + \beta_2\kappa_2^4) = \lambda(\beta_1\kappa_1^2 + \beta_2\kappa_2^2). \end{aligned} \tag{6.56b}$$

Since κ_1 and κ_2 must each give the same λ, we use (6.54) to obtain two additional equations

$$\lambda = \hat{a}_{-2}\kappa_1^{-2} + \hat{a}_{-1}\kappa_1^{-1} + \hat{a}_0 + \hat{a}_1\kappa_1, \tag{6.56c}$$

$$\lambda = \hat{a}_{-2}\kappa_2^{-2} + \hat{a}_{-1}\kappa_2^{-1} + \hat{a}_0 + \hat{a}_1\kappa_2. \tag{6.56d}$$

The system of (four) equations (6.56) has (five) unknowns $\beta_1, \beta_2, \kappa_1, \kappa_2, \lambda$. Since the equations (6.56a) and (6.56b) are linear and homogeneous in β_1 and β_2 and (6.56) are algebraic equations, they can be reduced to a single algebraic equation in a single variable (we choose κ_2), i.e.,

$$P(\kappa_2) = 0. \tag{6.57}$$

The roots of (6.57) are all possible values of κ_2 that may introduce boundary condition dependent eigenvalues. Each κ_2 is tested as follows: Substitute $\kappa = \kappa_2$ into (6.54). Solve the new equation for $\lambda(\kappa_2)$. Substitute $\lambda(\kappa_2)$ back into (6.54) and solve the resulting equation for $\kappa_1, \kappa_2, \kappa_3$. If the κ's satisfy (3.4) with (3.5b), i.e.,

$$|\kappa_1| \le |\kappa_2| < |\kappa_3|, \tag{6.58}$$

then $\lambda(\kappa_2)$ is a boundary condition dependent eigenvalue, otherwise it is not part of the asymptotic spectrum.

If the bandwidth of the matrix is large and the number of rows in the boundary condition matrix **B** becomes large, the algebraic reduction of the system, e.g., (6.57), becomes hopeless even for good symbolic systems such as MACSYMA. In these cases we recommend the less reliable but more efficient algorithm of §4.3.

An alternative solution procedure for (6.56) is the following. The system of equations, e.g., (6.56) is symmetric in the κ's so the algebra can be simplified by the introduction of the elementary symmetric functions. As an example, we consider the relatively simple right boundary problem for the matrix (1.7) and introduce the elementary symmetric functions [11]

$$y = \kappa_1 + \kappa_2 \tag{6.59}$$

and

$$x = \kappa_1 \kappa_2. \tag{6.60}$$

It can be shown that the system (6.56) reduces to

$$\begin{aligned} -c_{13}\left(\frac{\hat{a}_1}{\hat{a}_{-2}}\right)^2 x^5 + \frac{2c_{13}\hat{a}_{-1}\hat{a}_1}{\hat{a}_{-2}^2}x^4 + \left[-\frac{c_{13}\hat{a}_{-1}^2}{\hat{a}_{-2}^2} + \frac{(\hat{a}_1 - c_{12})\hat{a}_1}{\hat{a}_{-2}}\right]x^3 \\ + \left[c_{13} - \frac{(\hat{a}_1 - c_{12})\hat{a}_{-1}}{\hat{a}_{-2}}\right]x^2 + (\hat{a}_0 - c_{11})x - \hat{a}_{-2} = 0, \end{aligned} \tag{6.61}$$

where $c_{11} = -3\beta$, $c_{12} = 3\beta - 1/2$, and $c_{13} = -\beta$ and

$$y = (\hat{a}_1 x^2 - \hat{a}_{-1} x)/\hat{a}_{-2}. \tag{6.62}$$

It is not practical to proceed analytically, so for a particular set of matrix coefficients we solve (6.61) numerically for x and proceed as in the general Algorithm 4.2.

7. Eigenvector scaling. If conventional numerical eigenvalue packages, e.g., IMSL, EISPACK, etc., are used to compute the spectra for nonnormal Toeplitz matrices, large errors in the spectra may be encountered for matrices of relatively low order. For example if we choose the matrix

$$[-1/6, 1, \underline{-1/2}, -1/3] \tag{7.1}$$

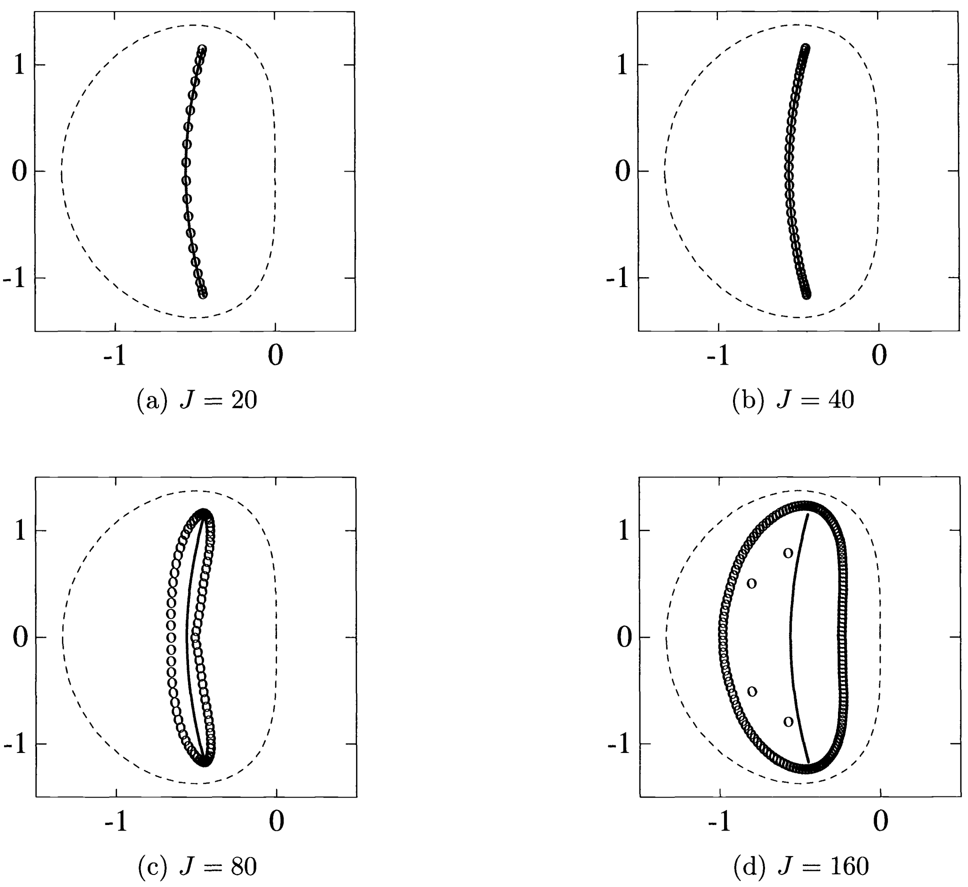


FIG. 7.1. *Numerically computed spectra with rounding errors* (o *symbol*) *for the unscaled Toeplitz matrix* $[-1/6, 1, \underline{-1/2}, -1/3]$. *Asymptotic Toeplitz spectra and asymptotic circulant spectra indicated by solid and dashed curves, respectively.*

and use the MATLAB routine *eig*($\mathbf{A}$) on an IRIS workstation to compute the spectrum, we obtain the sequence of spectra (for increasing matrix order) shown in Figs. 7.1(a) through 7.1(d). The Toeplitz matrix defined by (7.1) is the transpose of the Toeplitz matrix defined by (2.16).

The numerical spectra of the Toeplitz matrix (2.16), e.g., see Fig. 5.1, for $J = 80$ and 160, fall on the asymptotic spectra. In general, the numerical spectra of a nonnormal matrix $\mathbf{A}$ and its transpose $\mathbf{A}^T$ (for sufficiently large J) will differ in spite of the fact that the analytical spectra are identical. This phenomenon, which is a result of rounding errors, is discussed at the end of this section.

A nonnormal Toeplitz matrix (e.g., (7.1)) can have the property that the spectrum is very sensitive to perturbations of the matrix elements. For such matrices, Trefethen [12] and Reichel and Trefethen [8] suggest that an eigenvalue analysis may lead to incorrect conclusions, and it is more meaningful to analyze *pseudo*-eigenvalues. The *erroneous* eigenvalues (induced by rounding errors) plotted in Fig. 7.1 are a manifestation of the pseudospectra of the matrix (7.1) for increasing J. The numerical

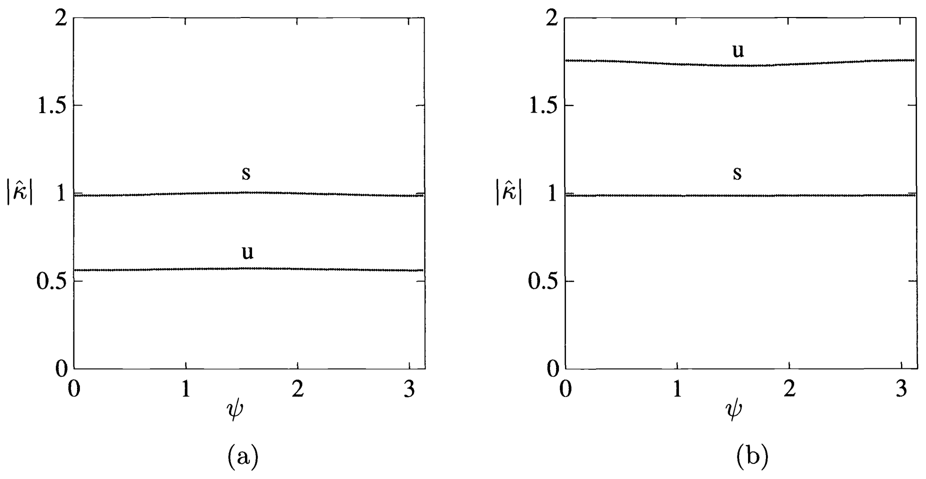


FIG. 7.2. *Modulus of* $\hat\kappa$ *vs* ψ_ℓ *for* (a) *matrix* (2.16) *unscaled,* u, *and scaled,* s, *and* (b) *the transpose matrix (7.1) unscaled,* u, *and scaled,* s.

error can be reduced by using higher numerical precision. However, for any numerical precision the qualitative behavior illustrated in Fig. 7.1 will always appear for sufficiently large J. Qualitatively, the numerical Toeplitz spectrum approaches the spectrum of its Toeplitz circulant cousin as $J \to \infty$.

The difficulty in numerically computing the eigenvalues of a nonnormal Toeplitz matrix is related to the exponential character of the eigenvectors. We illustrate this with an example by comparing the eigenvectors of the matrix (2.16) with the eigenvectors of its transpose (7.1). Since (2.16) is a Toeplitz matrix, the asymptotic spectrum is independent of the boundary conditions and the κ roots satisfy (6.23). In this example the moduli of the roots of equal modulus (4.1), $|\hat\kappa|$, are only weakly dependent on the angle ψ_ℓ as illustrated in Fig. 7.2. From (4.4) with coefficients given by (2.16) and $\psi = \pi/2$ one obtains $|\hat\kappa| = 1/\sqrt{3}$. Hence for the roots of equal modulus, one finds

$$|\kappa_1| = |\kappa_2| = |\hat\kappa| \approx 1/\sqrt{3}, \tag{7.2}$$

and consequently the moduli of the eigenvector elements behave as

$$|\phi_j| \approx (1/\sqrt{3})^j, \tag{7.3}$$

i.e., they decrease exponentially with increasing j.

For the transpose $\mathbf{A}^T$, defined by (7.1), one has

$$|\check\kappa_1| \le |\check\kappa_2| = |\check\kappa_3|, \tag{7.4}$$

where the check symbol indicates that the κ's in (6.23) and (7.4) are not the same. In fact, the κ's of (6.23) and (7.4) are reciprocals. Consequently, for $\mathbf{A}^T$ one has

$$|\check\kappa_2| = |\check\kappa_3| = |\hat\kappa| \approx \sqrt{3}, \tag{7.5}$$

and the moduli of the eigenvector elements

$$|\phi_j| \approx (\sqrt{3})^j \tag{7.6}$$

grow exponentially with increasing j.

A simple scaling attenuates the exponential behavior of the eigenvectors. Let $\mathbf{A}$ denote an arbitrary banded Toeplitz matrix. If the κ's of equal modulus (3.5a) are not on the unit circle, then either $\mathbf{A}$ or $\mathbf{A}^T$ will have exponentially growing eigenvectors. Assume that $\mathbf{A}$ has κ's of equal modulus (4.1) which are outside the unit circle. Define $\tilde{\kappa}$ to be

$$\tilde{\kappa} = \underset{0<\psi_\ell<\pi}{\text{mean}} (|\hat{\kappa}|). \tag{7.7}$$

For the matrix $\mathbf{A}$ we rescale the eigenvectors by

$$\tilde{\phi}_j = \frac{\phi_j}{\tilde{\kappa}^j}. \tag{7.8}$$

The scaling is effectively a normalization of the $\hat{\kappa}$'s so their moduli are approximately equal to unity.

The *scaling* similarity transformation is

$$\mathbf{S}^{-1} = \begin{bmatrix} 1/\tilde{\kappa} & & & & \\ & 1/\tilde{\kappa}^2 & & O & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & O & & 1/\tilde{\kappa}^{J-1} & \\ & & & & 1/\tilde{\kappa}^J \end{bmatrix}. \tag{7.9}$$

Hence the rescaled eigenvalue problem is

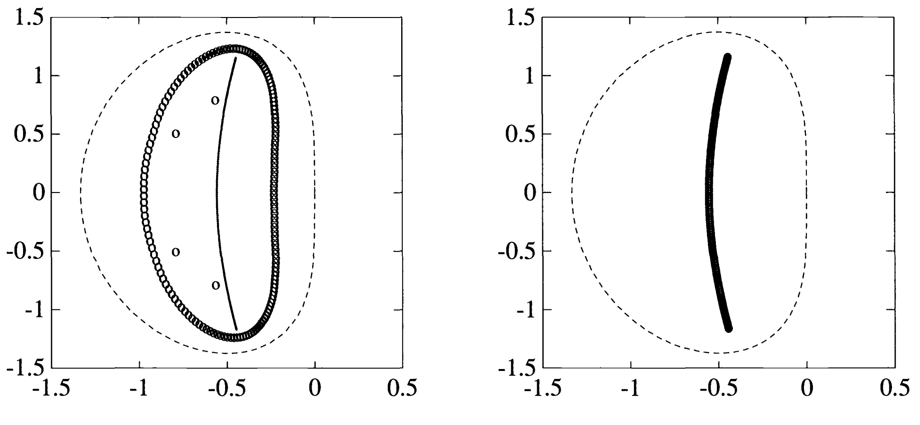
$$(\mathbf{S}^{-1}\mathbf{A}\mathbf{S})\mathbf{S}^{-1}\phi = \lambda \mathbf{S}^{-1}\phi. \tag{7.10}$$

If $a_{ij} = a_{j-i}$ are the nonzero elements of the Toeplitz matrix $\mathbf{A}$, the elements $\tilde{a}_{ij}$ of the rescaled matrix $\mathbf{S}^{-1}\mathbf{A}\mathbf{S}$ are

$$\tilde{a}_{ij} = a_{j-i}\tilde{\kappa}^{j-i}. \tag{7.11}$$

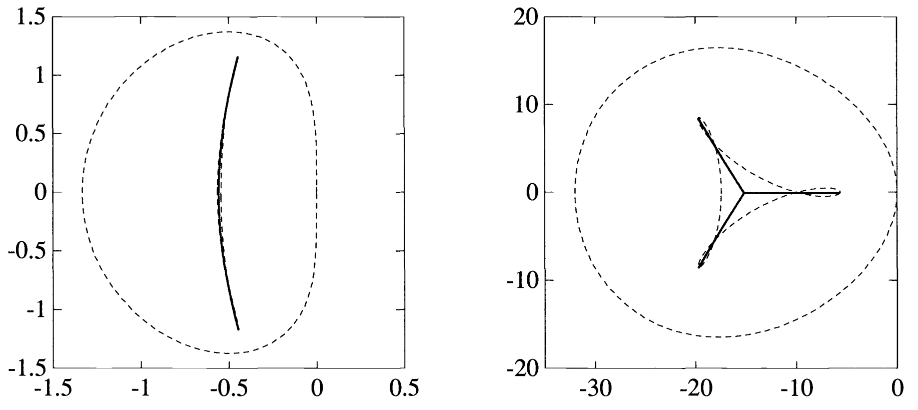
To numerically compute the spectrum of the rescaled eigenvalue problem, one simply replaces the matrix elements by (7.11). One should not make the *numerical* matrix multiplies indicated by $\mathbf{S}^{-1}\mathbf{A}\mathbf{S}$ because of potentially large rounding errors. The implementation of Algorithm 4.1 in the Appendix computes $\hat{\kappa}(\psi)$, which can be used to compute (7.7).

In Fig. 7.3 we compare the computed (MATLAB) spectrum of the scaled and unscaled Toeplitz matrix defined by (7.1). One can predict a bound on the numerical eigenvalue distribution of a Toeplitz matrix for large J. In Fig. 7.4(a) the outer dashed curve is the circulant cousin spectrum of the Toeplitz matrix defined by (7.1) and the inner dashed curve is the circulant cousin of the corresponding *scaled* Toeplitz matrix. The solid curve is the asymptotic Toeplitz spectrum. For the unscaled matrix the outer dashed curve of Fig. 7.4(a) is the bound on the numerically computed spectrum for large J. For the scaled matrix, the inner dashed curve is the asymptotic bound on the numerically computed spectrum. For the scaled matrix the circulant spectrum is tightly "wound" around the asymptotic Toeplitz spectrum and one can



(a) $J = 160$ (b) $J = 160$

FIG. 7.3. *Numerically computed spectra with rounding errors* (o *symbol*) *for* (a) *the unscaled Toeplitz matrix* $[-1/6, 1, \underline{-1/2}, -1/3]$, *and* (b) *the scaled matrix. Asymptotic Toeplitz spectra and asymptotic circulant spectra are indicated by solid and dashed curves, respectively.*



(a) $[-1/6, 1, \underline{-1/2}, -1/3]$ (b) $[-1.5, 16, \underline{-15}, 0, .5]$

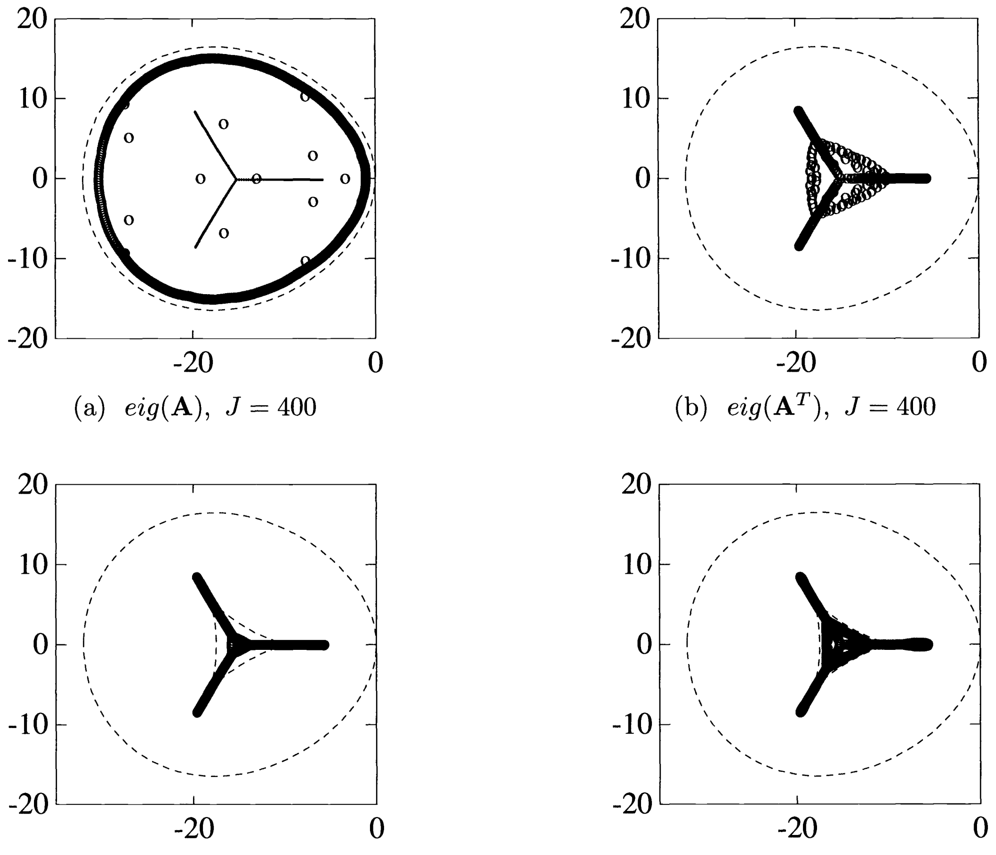
FIG. 7.4. *Asymptotic circulant spectra for unscaled* (*"outer" dashed curves*) *and scaled* (*"inner" dashed curves*) *matrices. Asymptotic Toeplitz spectra are indicated by solid curves.*

expect an accurately computed spectrum for large J. These bounds are illustrated by the numerical spectra shown in Fig. 7.3.

As a second example, we plot in Fig. 7.4(b) the spectra of the unscaled and scaled circulant cousins of the pentadiagonal Toeplitz matrix defined by

$$[-1.5, 16, \underline{-15}, 0, .5]. \tag{7.12}$$

The solid curve, i.e., the *propeller*, is the asymptotic Toeplitz spectrum. Numerically



(a) *eig*($\mathbf{A}$), $J = 400$ (b) *eig*($\mathbf{A}^T$), $J = 400$

(c) *eig*(*scaled*($\mathbf{A}$)), $J = 400$ (d) *eig*(*scaled*($\mathbf{A}^T$)), $J = 1000$

FIG. 7.5. *Numerically computed spectra with rounding errors* (o *symbols*) *for* (a) $\mathbf{A} = [-1.5, 16, \underline{-15}, 0, .5]$, (b) $\mathbf{A}^T$, (c) *scaled* $\mathbf{A}$, $J = 400$, *and* (d) *scaled* $\mathbf{A}^T$, $J = 1000$.

computed (MATLAB) spectra of the Toeplitz matrix and its transpose are plotted in Figs. 7.5(a) and 7.5(b) for $J = 400$. The spectra for both $\mathbf{A}$ and $\mathbf{A}^T$ are erroneous for large J. The inner dashed curve of Fig. 7.4(b) is the asymptotic bound on the numerical eigenvalue distribution of the scaled matrix for large J. Numerical eigenvalue computations for the scaled matrix are shown in Figs. 7.5(c) and 7.5(d).

One can also predict how well the scaling will work. If a graph of $|\hat{\kappa}|$ vs ψ_ℓ (see Algorithm 4.1) for the unscaled matrix is a single flat band as in Fig. 7.2, then the scaling will yield a tight bound for the circulant cousin around the asymptotic Toeplitz spectrum as illustrated in Fig. 7.4(a). On the other hand, if $|\hat{\kappa}|$ vs ψ_ℓ forms multiple and/or nonuniform bands as illustrated in Fig. 7.6, then one has multiple scales and a simple scaling will not yield a closely wound circulant cousin spectrum as illustrated in Fig. 7.4(b).

If one starts with a quasi-Toeplitz matrix, then the appropriate scaling is the same as for its pure Toeplitz cousin.

8. The generalized eigenvalue problem. If Padé type difference approxima-

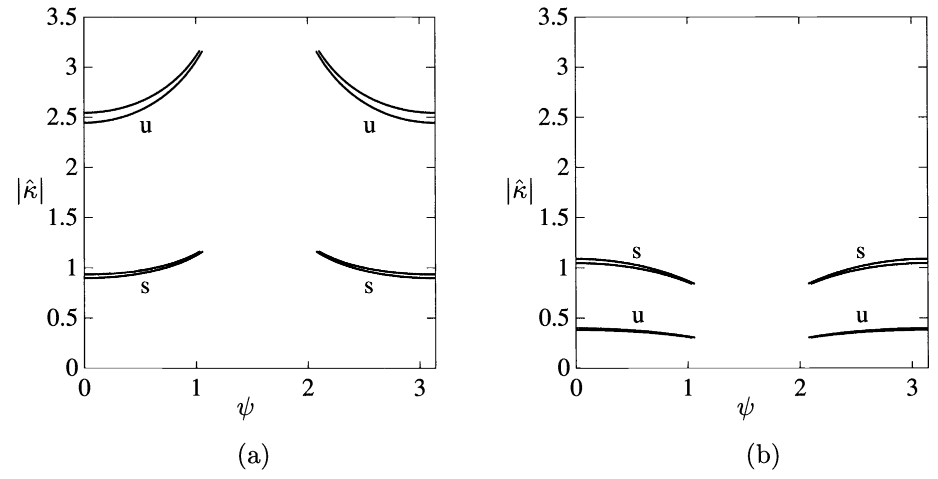


FIG. 7.6. *Modulus of $\hat\kappa$ vs ψ_ℓ for* (a) *matrix* $[-1.5, 16, \underline{-15}, 0, .5]$ *unscaled,* u, *and scaled,* s, *and* (b) *the transpose matrix unscaled,* u, *and scaled,* s.

tions are used to approximate differential equations the associated stability analysis leads to a generalized eigenvalue problem. In addition, the analysis of implicit time integration schemes leads to a generalized eigenvalue problem. Let $\mathbf{A}$ and $\mathbf{B}$ represent $J \times J$ banded Toeplitz matrices. The generalized eigenvalue problem is defined by

$$\mathbf{A}\boldsymbol{\phi} = \lambda \mathbf{B}\boldsymbol{\phi}, \tag{8.1}$$

where $\boldsymbol{\phi}$ is the eigenvector and λ is the eigenvalue. Here we assume that the matrix $\mathbf{A}$ is defined by the sequence $[a_i, -p_A \le i \le q_A]$ and $\mathbf{B}$ is defined by the sequence $[b_i, -p_B \le i \le q_B]$, where p_A, q_A, p_B, q_B are positive integers. The eigenvector $\boldsymbol{\phi}$ is given by

$$\boldsymbol{\phi}^T = [\phi_1, \phi_2, \cdots, \phi_{J-1}, \phi_J]. \tag{8.2}$$

The eigenvalue problem (8.1) is equivalent to the set of homogeneous equations

$$\sum_{m=-p_A}^{q_A} a_m \phi_{j+m} = \lambda \sum_{m=-p_B}^{q_B} b_m \phi_{j+m}, \quad j = 1, 2, \cdots, J \tag{8.3}$$

with $p = \max[p_A, p_B]$ homogeneous boundary conditions at the left boundary and $q = \max[q_A, q_B]$ homogeneous boundary conditions at the right boundary

$$\phi_{-m} = 0, \quad m = 0, 1, \cdots, p-1, \tag{8.4a}$$

$$\phi_{J+m} = 0, \quad m = 1, 2, \cdots, q. \tag{8.4b}$$

Equation (8.3) is a $(p+q)$th order difference equation for ϕ_j and we look for a solution of the form

$$\phi_j = \kappa^j, \tag{8.5}$$

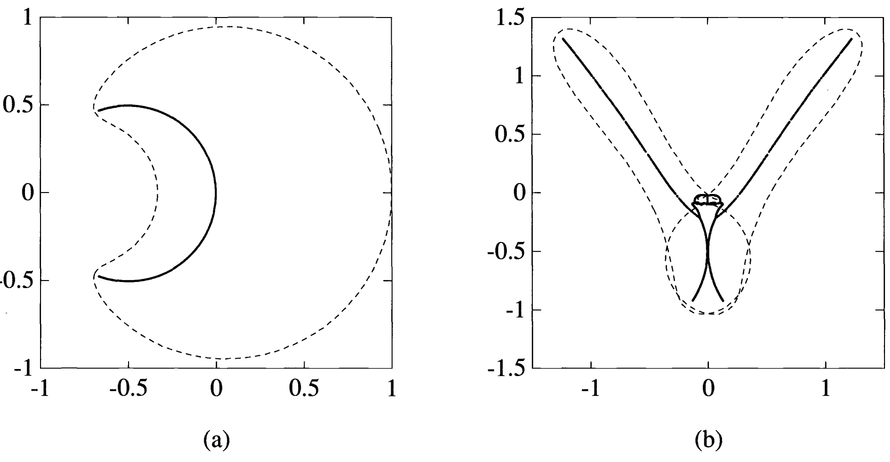


FIG. 8.1. *Spectra for the generalized Toeplitz eigenvalue problem* (8.1), (a) $\mathbf{A} = [2, \underline{0}, -1]$ *and* $\mathbf{B} = [-2, \underline{2}, 1]$ *and* (b) $\mathbf{A} = [.38, .13, -.43, .15, \underline{.25}, -.06, .27, -.02, -.26]$ *and* $\mathbf{B} = [.28i, .38i, .50i, .07i, \underline{.95i}, .99i, .49i, .27i, .09i]$.

where κ is a complex constant. Substitution of (8.5) into (8.3) yields

$$\lambda \sum_{m=-p_B}^{q_B} b_m \kappa^m = \sum_{m=-p_A}^{q_A} a_m \kappa^m. \tag{8.6}$$

This is an algebraic equation of degree $p+q$ in the unknown κ. The general solution of (8.3) is (2.7). The constants β are determined by inserting (2.7) into the boundary conditions (8.4). If the bandwidth is three, i.e., $p = q = 1$, the eigenvalue problem for the pure Toeplitz problem can be solved analytically [16]. However, if the bandwidth is greater than three, the problem is analytically intractable.

The eigenvalue problem for the circulant cousin of (8.1) is (8.3) with periodic boundary conditions (2.12). Insertion of (8.5) into (2.12) yields

$$\kappa^J = 1. \tag{8.7}$$

Hence the κ's are the J roots of unity given by (2.14) and the circulant spectrum is obtained by substituting (2.14) into (8.6)

$$\lambda_\ell \sum_{m=-p_B}^{q_B} b_m e^{im\theta_\ell} = \sum_{m=-p_B}^{q_B} a_m e^{im\theta_\ell}. \tag{8.8}$$

The partition of the asymptotic Toeplitz spectrum is the same (§3) as for the standard eigenvalue problem. The algorithm follows that of §4.1. To construct the polynomial for κ's of equal modulus we use (4.1). Since κ_a and κ_b (defined by 4.1) must each give the same value of λ when inserted in (8.6), we can eliminate λ and obtain a polynomial in $\hat{\kappa}$ with coefficients which are functions of the Toeplitz matrix coefficients and ψ_ℓ. In particular,

$$\lambda(\kappa_a) \sum_{m=-p_B}^{q_B} b_m \hat{\kappa}^m e^{im\psi_\ell} = \sum_{m=-p_A}^{q_A} a_m \hat{\kappa}^m e^{im\psi_\ell}, \tag{8.9a}$$

$$\lambda(\kappa_b) \sum_{m=-p_B}^{q_B} b_m \hat{\kappa}^m e^{-im\psi_\ell} = \sum_{m=-p_A}^{q_A} a_m \hat{\kappa}^m e^{-im\psi_\ell}, \tag{8.9b}$$

and since

$$\lambda(\kappa_a) = \lambda(\kappa_b), \tag{8.10}$$

one obtains, after some algebraic simplification,

$$\sum_{m=-p_A}^{q_A} \sum_{n=-p_B}^{q_B} a_m b_n \hat{\kappa}^{(m+n)} \sin[(m-n)\psi_\ell] = 0. \tag{8.11}$$

The algorithm follows that of (i) and (ii) in §4.1. The implementation of the generalized eigenvalue algorithm is a straightforward modification of the standard eigenvalue algorithm (Appendix).

As an example of the $\hat{\kappa}$ polynomial (8.11), if we choose $\mathbf{A}$ and $\mathbf{B}$ to be the quadridiagonal matrices represented by the sequences $[a_{-2}, a_{-1}, a_0, a_1]$ and $[b_{-2}, b_{-1}, b_0, b_1]$, the equation for the κ's of equal modulus is

$$\begin{aligned}(a_1 b_0 - a_0 b_1)\hat{\kappa}^4 &+ 2(a_1 b_{-1} - a_{-1} b_1)\cos(\psi_\ell)\hat{\kappa}^3 \\ &+ [(a_0 b_{-1} - a_{-1} b_0) + (a_1 b_{-2} - a_{-2} b_1)(4\cos^2(\psi_\ell) - 1)]\hat{\kappa}^2 \\ &+ 2(a_0 b_{-2} - a_{-2} b_0)\cos(\psi_\ell)\hat{\kappa} + (a_{-1} b_{-2} - a_{-2} b_{-1}) = 0.\end{aligned} \tag{8.12}$$

Note that (6.4) and the "transpose" of (6.25) are easily obtained as special cases of (8.12). As examples of the spectra of the generalized eigenvalue problem we choose two examples. The spectra for the Toeplitz and the associated circulant Toeplitz matrices are plotted in Fig. 8.1

9. Concluding remarks.

The algorithms presented in §4 provide a practical and accurate method for obtaining the asymptotic spectra of banded Toeplitz and quasi-Toeplitz matrices. They are especially useful for nonnormal matrices where conventional algorithms may give erroneous results. The extension of the algorithms for the generalized eigenvalue problem for Toeplitz matrices was presented in §8. The asymptotic analysis also provides a similarity transformation, §7, which can increase the spectrum accuracy if conventional eigenvalue algorithms (finite J) are employed.

Acknowledgments. We are grateful to Dennis Jespersen, Robert Schreiber, Nick Trefethen, and an anonymous reviewer for comments and suggestions that improved our manuscript.

Appendix. MATLAB Toeplitz eigenvalue algorithm.

```
function[teig, spsi, sakap] = toeasy(c, r, nn);
%Find the asymptotic (size approaches infinity) spectrum 'teig'
%and κ̂(ψ) 'sakap' and ψ 'spsi' (see Algorithm 4.1)
%for a banded Toeplitz matrix with first column 'c' and first row
%'r' (only banded part of column and row) 'nn' is a measure of the
% number of points computed on the spectrum
diag = c(1);
lr = length(r);
```

```
nr = lr − 1;
lc = length(c);
nc = lc − 1;
ct = c(2 : lc);
rt = r(2 : lr);
mv = nr : −1 : −nc;
pc0 = [fliplr(rt) 0 ct];
teig = [1 : (nc + nr) ∗ nn];
spsi = [1 : (nc + nr) ∗ nn];
sakap = [1 : (nc + nr) ∗ nn];
eigct = 0;
epst = 1000 ∗ eps;
for j = 1 : nn;
 psi = j ∗ pi/(nn + 1);
 mvs = sin(mv ∗ psi);
 pe = pc0. ∗ mvs;
 nroot = nc + nr;
 if (pe(2) ∼= 0);
  if(abs(pe(1)/pe(2)) < 1000 ∗ eps);
   pe = pe(2 : nroot + 1);
   nroot = nroot − 1;
  end;
 end;
 ke = roots(pe);
 for k = 1 : nroot;
 if ke(k) ∼= 0;
  kap = ke(k) ∗ exp(i ∗ psi);
  lam = −(kap. ∧ mv) ∗ pc0.′;
  pc = [fliplr(rt) lam ct];
  kc = roots(pc);
  akap = abs(kap);
  akc = abs(kc);
  skap = sort(akc);
  test = abs((skap(nc) − skap(nc + 1))/(skap(nc) + skap(nc + 1)));
  if test < epst&skap(nc) > akap ∗ (1 − epst)&skap(nc) < akap ∗ (1 + epst);
   eigct = eigct + 1;
   teig(eigct) = −(lam − diag);
   spsi(eigct) = psi;
   sakap(eigct) = akap;
  end;
 end;
end;
end;
teig = teig(1 : eigct);
spsi = spsi(1 : eigct);
sakap = sakap(1 : eigct);
```

REFERENCES

[1] P. J. Davis, *Circulant Matrices*, John Wiley, New York, 1979.

[2] S. K. Godunov and V. S. Ryabenkii, *Special stability criteria of boundary value problems for non-selfadjoint difference equations*, Russ. Math. Surv., 18 (1963), pp. 1–12.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Second Edition, Johns Hopkins University Press, Baltimore, 1989.

[4] R. M. Gray, *On the asymptotic eigenvalue distribution of Toeplitz matrices*, IEEE Trans. Info. Theory, 18 (1972), pp. 725–730.

[5] U. Grenander and G. Szegö, *Toeplitz Forms and Their Applications*, Univ. of California Press, Berkeley, 1958.

[6] B. Gustafsson, H.-O. Kreiss, and A. Sundström, *Stability theory of difference approximations for mixed initial boundary value problems*. II, Math. Comp., 26 (1972), pp. 649–686.

[7] H.-O. Kreiss, *Stability theory for difference approximations of mixed initial boundary value problems*. I, Math. Comp., 22 (1968), pp. 703–714.

[8] L. Reichel and L. N. Trefethen, *Eigenvalues and pseudo-eigenvalues of Toeplitz matrices*, Linear Algebra Appl., to appear..

[9] P. Schmidt and F. Spitzer, *The Toeplitz matrices of an arbitrary Laurent polynomial*, Math. Scand., 8 (1960), pp. 15–38.

[10] G. D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Second Edition, Oxford University Press, Oxford, 1978.

[11] J. C. Strikwerda, *Initial boundary problems for the method of lines*, J. Comput. Phys., 34 (1980), pp. 94–107.

[12] L. N. Trefethen, *Pseudospectra of matrices*, in Numerical Analysis 1991, Proc. 14th Dundee Conf., D.F. Griffiths and G. A. Watson, eds., Pittman Research Notes in Mathematics Series 260, Longman Scientific and Technical, Harlow, 1992, pp. 234–266.

[13] W. F. Trench, *On the eigenvalue problem for Toeplitz band matrices*, Linear Algebra Appl., 64 (1985), pp. 199-214.

[14] J. L. Ullman, *A problem of Schmidt and Spitzer*, Bull. Amer. Math. Soc., 73 (1967), pp. 883–885.

[15] R. F. Warming and R. M. Beam, *An eigenvalue analysis of finite-difference approximations for hyperbolic* IBVPs II: *the auxiliary Dirichlet problem*, in Third International Conference on Hyperbolic Problems, Vol. II, B. Engquist and B. Gustafsson, eds., Studentlitteratur, Lund, 1991, pp. 923–937.

[16] ———, *The generalized eigenvalue problem for tridiagonal Toeplitz matrices*, to appear.

SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 1007–1012, July 1993

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific 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 NEW PREPROCESSING ALGORITHM FOR THE COMPUTATION OF THE GENERALIZED SINGULAR VALUE DECOMPOSITION*

ZHAOJUN BAI† AND HONGYUAN ZHA‡

Abstract. In this note a new algorithm is proposed for the preprocessing phase of Paige's algorithm for computing the generalized singular value decomposition (GSVD). This new algorithm substantially reduces the complexity of Paige's algorithm and makes it much easier to implement. It is also proved that the preprocessing phase is backward stable and a numerical example is demonstrated.

Key words. generalized singular value decomposition, URV decomposition

AMS subject classification. 65F30

CR classification. G1.3

1. Introduction. This note is concerned with the numerical computation of the generalized singular value decomposition (GSVD) of two matrices having the same number of columns. The GSVD was first proposed by Van Loan [9]. Like the singular value decomposition (SVD) of one matrix, the GSVD of matrix pairs is a very useful tool in many numerical linear algebra problems. The following formulation of the GSVD is due to Paige and Saunders [5], and is more suitable for numerical computation.

THEOREM 1.1. *Given a matrix pair (A, B) with $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times n}$, there exist orthogonal matrices U, V, and Q such that*

$$U^TAQ = \Sigma_A(\,0 \;\; R\,), \qquad V^TBQ = \Sigma_B(\,0 \;\; R\,), \tag{1}$$

where R is $k \times k$ nonsingular upper triangular with $k = \operatorname{rank}(A^T, B^T)^T$, and Σ_A and Σ_B are $m \times k$ and $p \times k$ diagonal matrices such that

$$\Sigma_A^T\Sigma_A + \Sigma_B^T\Sigma_B = I.$$

Paige's algorithm for computing the above-mentioned decomposition consists of two phases: (1) reducing the given matrix pairs to upper triangular (or trapezoidal) forms by orthogonal transformations, which is designated as the preprocessing phase; (2) implicitly applying the Kogbetliantz algorithm to find the GSVD of two triangular matrices [6]. The existing preprocessing procedure may result in an *irregular* triangular pair, which gives rise to several different cases in the second phase and makes the numerical implementation quite complicated [2]. In this note the authors propose a new preprocessing algorithm that essentially leaves only one case in the next phase. More precisely, the authors reduce the problem to the computation of the GSVD of a matrix pair (A, B), where A and B are upper triangular and B nonsingular, which is called a *regular* matrix pair.[1] Our current interest in developing algorithms for computing the GSVD stems

*Received by the editors March 17, 1992; accepted for publication (in revised form) October 23, 1992.
†Department of Mathematics, University of Kentucky, Lexington, Kentucky 40506.
‡Scientific Computing and Computational Mathematics, Stanford University, Stanford, California 94305.
[1]The generalized singular values of a regular matrix pair are all finite.

from Bai's involvement in the linear algebra software package LAPACK [1]. Paige's algorithm for computing the GSVD will be included in the future release of LAPACK. For an alternative GSVD algorithm via the CS decomposition, the reader is referred to [7] and [10].

2. A new preprocessing algorithm. The purpose of the preprocessing phase is to reduce the given matrix pair (A, B) to a condensed form, so that the implicit Kogbetliantz algorithm can be applied. Our approach is to extract a regular matrix pair from (A, B) by applying orthogonal transformations to the individual matrices A and B. A similar idea was also used in [11] for computing the *restricted singular value decomposition* of matrix triplets. The algorithm consists of three steps. The transformation from step k to step $k+1$ is denoted as

$$\begin{pmatrix} A^{(k+1)} \\ B^{(k+1)} \end{pmatrix} = \begin{pmatrix} (U^{(k)})^T A^{(k)} (Q^{(k)}) \\ (V^{(k)})^T B^{(k)} (Q^{(k)}) \end{pmatrix},$$

where $U^{(k)}, V^{(k)}$, and $Q^{(k)}$ are orthogonal; $A^{(k)}$ and $B^{(k)}$ are the transformed A and B at step k with initial values

$$A^{(0)} = A, \qquad B^{(0)} = B.$$

For brevity, in the following description of the algorithm, we only specify the transformed A and B at each step; the corresponding orthogonal transformation matrices can be easily constructed.

Before we proceed, we need to recall a matrix decomposition, which will be the building block of our preprocessing algorithm: for any $m \times n$ matrix A, there exist orthogonal matrices U and V such that A can be decomposed as

$$A = U \begin{pmatrix} 0 & R \\ 0 & 0 \end{pmatrix} V^T,$$

where R is a nonsingular upper triangular matrix. The decomposition is called the *complete orthogonal decomposition*, or simply the *URV decomposition* [3]. Now we are ready to present the preprocessing algorithm.

Step 1. Compute the URV decomposition of B such that

$$\begin{pmatrix} A^{(1)} \\ B^{(1)} \end{pmatrix} = \left(\begin{array}{cc} A_{11}^{(1)} & A_{12}^{(1)} \\ \hline 0 & B_{12}^{(1)} \\ 0 & 0 \end{array} \right),$$

where $B_{12}^{(1)}$ is upper triangular and nonsingular.

Step 2. Compute the URV decomposition of $A_{11}^{(1)}$ (if $A_{11}^{(1)}$ is not empty) and update $A_{12}^{(1)}$, so that we have

$$\begin{pmatrix} A^{(2)} \\ B^{(2)} \end{pmatrix} = \left(\begin{array}{ccc} 0 & A_{12}^{(2)} & A_{13}^{(2)} \\ 0 & 0 & A_{23}^{(2)} \\ \hline 0 & 0 & B_{13}^{(2)} \\ 0 & 0 & 0 \end{array} \right),$$

where $A_{12}^{(2)}$ is upper triangular and nonsingular, and $B_{13}^{(2)} = B_{12}^{(1)}$.

Step 3. Compute the QR decomposition of $A_{23}^{(2)}$, such that

$$\begin{pmatrix} A^{(3)} \\ B^{(3)} \end{pmatrix} = \left(\begin{array}{ccc} 0 & A_{12}^{(3)} & A_{13}^{(3)} \\ 0 & 0 & A_{23}^{(3)} \\ \hline 0 & 0 & B_{13}^{(3)} \\ 0 & 0 & 0 \end{array} \right),$$

where $A_{12}^{(3)} = A_{12}^{(2)}$, $A_{13}^{(3)} = A_{13}^{(2)}$, and $B_{13}^{(3)} = B_{13}^{(2)}$. For $A_{23}^{(3)}$, we need to distinguish two cases. Let $A_{23}^{(3)} \in R^{s\times r}$.

1. If $s \geq r$, then

$$A_{23}^{(3)} = \begin{pmatrix} \tilde{A}_{23}^{(3)} \\ 0 \end{pmatrix},$$

where $\tilde{A}_{23}^{(3)}$ is $r \times r$ upper triangular.

2. If $s < r$, then the first s columns of $A_{23}^{(3)}$ is an upper triangular matrix. We append $r - s$ rows of zeros to $A_{23}^{(3)}$ and denote the resulting matrix $\tilde{A}_{23}^{(3)}$, which is also an upper triangular matrix.

In either case, we end up with a matrix pair $(\tilde{A}_{23}^{(3)}, B_{13}^{(3)})$, where $\tilde{A}_{23}^{(3)}$ and $B_{13}^{(3)}$ are square upper triangular and of the same size; moreover, $B_{13}^{(3)}$ is nonsingular. Therefore, computing the GSVD of $(\tilde{A}_{23}^{(3)}, B_{13}^{(3)})$ is equivalent to computing the SVD of $\tilde{A}_{23}^{(3)}(B_{13}^{(3)})^{-1}$. Using the implicit Kogbetliantz algorithm [6], [2], we can find three orthogonal matrices U_1, V_1, and Q_1 such that

$$U_1^T \tilde{A}_{23}^{(3)} (B_{13}^{(3)})^{-1} V_1 = \text{diagonal},$$

and

$$U_1^T \tilde{A}_{23}^{(3)} Q_1 = \Sigma_1 R_1, \qquad V_1^T B_{13}^{(3)} Q_1 = \Sigma_2 R_1, \tag{2}$$

where Σ_1 and Σ_2 are diagonal, R_1 is upper triangular, and

$$\Sigma_1^T \Sigma_1 + \Sigma_2^T \Sigma_2 = I.$$

Combining Steps 1–3 and the results from the implicit Kogbetliantz algorithm (2), and accumulating the corresponding orthogonal transformations, we have

$$U^T A Q = \Sigma_A \begin{pmatrix} 0 & R \end{pmatrix} \quad \text{and} \quad V^T B Q = \Sigma_B \begin{pmatrix} 0 & R \end{pmatrix}, \tag{3}$$

where

$$\Sigma_A = \begin{pmatrix} I & 0 \\ 0 & \Sigma_1 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \Sigma_B = \begin{pmatrix} 0 & \Sigma_2 \\ 0 & 0 \end{pmatrix},$$

and the upper triangular matrix R is given by

$$R = \begin{pmatrix} A_{12}^{(3)} & A_{13}^{(3)} Q_1 \\ 0 & R_1 \end{pmatrix}.$$

Equation (3) gives the desired GSVD of A and B in Theorem 1.1.

3. Backward stability and numerical example. In this section, we first discuss the numerical stability of the new preprocessing algorithm and then present one example as a demonstration. The crux of the numerical stability of the new preprocessing algorithm is how to stably compute the URV decomposition. In the current implementation, this is done by first applying the QR decomposition with column pivoting, and then squeezing the resulting trapezoidal form into upper triangular form by applying a sequence of Householder transformations. Let quantities with an overbar denote computed quantities. From the standard backward error analysis of the QR decomposition [3], we know that for the above preprocessing algorithm, the computed $\bar{A}^{(3)}$ and $\bar{B}^{(3)}$ satisfy

$$\bar{U}^T(A+E)\bar{Q} = \bar{A}^{(3)}, \qquad \bar{V}^T(B+F)\bar{Q} = \bar{B}^{(3)},$$

where

$$\|E\|_F \leq \tau\|A\|_2, \qquad \|F\|_F \leq \tau\|B\|_2,$$

and where τ is a user-specified tolerance value, which is used in the QR decomposition with column pivoting to determine the effective numerical rank of the matrix. We also mention that there exist more sophisticated algorithms for computing the URV decomposition, for example, the one proposed by Hanson and Lawson [4] and Stewart [8].

To conclude this note, we apply the new preprocessing algorithm and the implicit Kogbetliantz algorithm to a numerical example.[2] Our emphasis here is the backward stability of the preprocessing algorithm. The matrix pair (A, B) is given as follows:

$$A = \begin{pmatrix} 1 & 2 & 3 & 1 & 5 \\ 0 & 3 & 2 & 0 & 2 \\ 1 & 0 & 2 & 1 & 0 \\ 0 & 2 & 3 & 0 & -1 \\ 1 & 0 & 2 & 1 & 1 \\ 0 & 2 & 1 & 0 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & -2 & 2 & 1 & 1 \\ 0 & 3 & 0 & 0 & 0 \\ 1 & -2 & 2 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 2 & -4 & 4 & 2 & 2 \\ 1 & 3 & 2 & 1 & 1 \end{pmatrix}.$$

After the three steps of preprocessing algorithm with the tolerance value set as $\epsilon_M\|A\|_F$ and $\epsilon_M\|B\|_F$ for the matrices A and B, respectively, we have

$$\bar{A}^{(3)} = \left(\begin{array}{c|cc|cc} 0 & 3.6017\text{E}+00 & -1.7136\text{E}+00 & 3.3819\text{E}-01 & 1.8012\text{E}+00 \\ 0 & 0 & -2.6088\text{E}+00 & 4.4448\text{E}+00 & 4.9805\text{E}+00 \\ \hline 0 & 0 & 0 & -1.0628\text{E}+00 & 6.4454\text{E}-02 \\ 0 & 0 & 0 & 0 & 4.2699\text{E}+00 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right),$$

[2]For brevity, only five decimal digits are displayed for all the data, though all the results are obtained using our FORTRAN routines run in double precision on an HP Apollo workstation with machine precision $\epsilon_M \approx 2.2204 \times 10^{-16}$.

$$\bar{B}^{(3)} = \left(\begin{array}{c|cc|cc} 0 & 0 & 0 & -6.7823\text{E}+00 & 3.5109\text{E}+00 \\ 0 & 0 & 0 & 0 & 6.0559\text{E}+00 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right),$$

where the partitioning of the matrix pair $(\bar{A}^{(3)}, \bar{B}^{(3)})$ corresponds to that in Step 3 of the preprocessing algorithm. At the second phase, we use the scheme proposed by Bai and Demmel [2] to compute the GSVD of the (2,3) block of $\bar{A}^{(3)}$ and the (1,3) block of $\bar{B}^{(3)}$; we obtain

$$\begin{pmatrix} -1.0628\text{E}+00 & 6.4454\text{E}-02 \\ 0 & 4.2699\text{E}+00 \end{pmatrix} = \begin{pmatrix} 1.5379\text{E}-01 & 0 \\ 0 & 5.7885\text{E}-01 \end{pmatrix} \bar{R}_1$$

and

$$\begin{pmatrix} -6.7823\text{E}+00 & 3.5109\text{E}+00 \\ 0 & 6.0559\text{E}+00 \end{pmatrix} = \begin{pmatrix} 9.8810\text{E}-01 & 0 \\ 0 & 8.1544\text{E}-01 \end{pmatrix} \bar{R}_1,$$

where

$$\bar{R}_1 = \begin{pmatrix} -6.9692\text{E}+00 & 3.5064\text{E}+00 \\ 0 & 7.3144\text{E}+00 \end{pmatrix}.$$

Combining the above two phases, we have

$$\bar{U}^T(A+E)\bar{Q} = \bar{\Sigma}_A(0 \;\; \bar{R}), \qquad \bar{V}^T(B+F)\bar{Q} = \bar{\Sigma}_B(0 \;\; \bar{R}), \tag{4}$$

where

$$\bar{\Sigma}_A = \begin{pmatrix} 1.0000\text{E}+00 & 0 & 0 & 0 \\ 0 & 1.0000\text{E}+00 & 0 & 0 \\ 0 & 0 & 1.5379\text{E}-01 & 0 \\ 0 & 0 & 0 & 5.7885\text{E}-01 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\bar{\Sigma}_B = \begin{pmatrix} 0 & 0 & 9.8810\text{E}-01 & 0 \\ 0 & 0 & 0 & 8.1544\text{E}-01 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

and

$$\bar{R} = \begin{pmatrix} 3.6017\text{E}+00 & -1.7136\text{E}+00 & 2.8436\text{E}-01 & 1.8104\text{E}+00 \\ 0 & -2.6088\text{E}+00 & 4.2944\text{E}+00 & 5.1107\text{E}+00 \\ 0 & 0 & -6.9692\text{E}+00 & 3.5064\text{E}+00 \\ 0 & 0 & 0 & 7.3144\text{E}+00 \end{pmatrix},$$

with the backward errors of the computed decomposition

$$\|E\|_F = \|\bar{U}^{\mathrm{T}}A\bar{Q} - \bar{\Sigma}_A\bar{R}\|_F = 4.5118\text{E}-15 \approx \epsilon_{\mathrm{M}}\|\mathrm{A}\|_{\mathrm{F}},$$

$$\|F\|_F = \|\bar{V}^{\mathrm{T}}B\bar{Q} - \bar{\Sigma}_B\bar{R}\|_F = 5.6621\text{E}-15 \approx \epsilon_{\mathrm{M}}\|\mathrm{B}\|_{\mathrm{F}}.$$

The computed orthogonal matrices $\bar{U}$, $\bar{V}$, and $\bar{Q}$ are orthogonal within machine precision. The decomposition in (4) gives the desired GSVD of A and B in Theorem 1.1.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[2] Z. Bai and J. W. Demmel, *Computing the generalized singular value decomposition*, Report UCB/CSD 91/645, Computer Science Division, University of California, Berkeley, 1991.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.

[4] R. J. Hanson and C. L. Lawson, *Extensions and applications of the Householder algorithm for solving linear least squares problems*, Math. Comp., 22 (1969), pp. 787–812.

[5] C. C. Paige and M. A. Saunders, *Towards a generalized singular value decomposition*, SIAM J. Numer. Anal., 18 (1981), pp. 398–405.

[6] C. C. Paige, *Computing the generalized singular value decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1126–1146.

[7] G. W. Stewart, *Computing the CS-decomposition of a partitioned orthonormal matrix*, Numer. Math., 40 (1982), pp. 297–306.

[8] ———, *An updating algorithm for subspace tracking*, CS-TR 2494, Department of Computer Science, Univ. of Maryland, College Park, 1990; IEEE Trans. on Signal Process, to appear.

[9] C. F. Van Loan, *Generalizing the singular value decomposition*, SIAM J. Numer. Anal., 13 (1976), pp. 76–83.

[10] ———, *Computing the CS and the generalized singular value decomposition*, Numer. Math., 46 (1985), pp. 479–491.

[11] H. Zha, *A numerical algorithm for computing restricted singular value decomposition of matrix triplets*, Linear Algebra Appl., 168 (1992), pp. 1–25.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1013–1019, September 1993

FAST ITERATIVE SOLVERS FOR TOEPLITZ-PLUS-BAND SYSTEMS*

RAYMOND H. CHAN† AND KWOK-PO NG†

Abstract. The authors consider the solutions of Hermitian Toeplitz-plus-band systems $(A_n + B_n)x = b$, where A_n are n-by-n Toeplitz matrices and B_n are n-by-n band matrices with bandwidth independent of n. These systems appear in solving integrodifferential equations and signal processing. However, unlike the case of Toeplitz systems, no fast direct solvers have been developed for solving them. In this paper, the preconditioned conjugate gradient method with band matrices as preconditioners is used. The authors prove that if A_n is generated by a nonnegative piecewise continuous function and B_n is positive semidefinite, then there exists a band matrix C_n, with bandwidth independent of n, such that the spectra of $C_n^{-1}(A_n + B_n)$ are uniformly bounded by a constant independent of n. In particular, we show that the solution of $(A_n+B_n)x = b$ can be obtained in $O(n \log n)$ operations.

Key words. Toeplitz matrix, band matrix, generating function, preconditioned conjugate gradient method

AMS subject classifications. 65F10, 65F15

1. Introduction.

In this paper, we consider the solution of systems of the form $(A_n + B_n)x = b$, where A_n is an n-by-n Hermitian Toeplitz matrix (i.e., the entries of A_n are the same along its diagonals) and B_n is an n-by-n Hermitian band matrix with bandwidth independent of n. These systems appear in solving Fredholm integro-differential equations of the form

$$L\{x(\theta)\} + \int_\alpha^\beta K(\phi - \theta)x(\phi)d\phi = b(\theta).$$

Here $x(\theta)$ is the unknown function to be found, $K(\theta)$ is a convolution kernel, L is a differential operator, and $b(\theta)$ is a given function. After discretization, K will lead to a Toeplitz matrix, L a band matrix, and $b(\theta)$ the right-hand side vector; see Delves and Mohamed [6, p. 343]. Toeplitz-plus-band matrices also appear in signal processing literature and have been referred to as peripheral innovation matrices; see Carayannis, Kalouptsidis, and Manolakis [2].

For Toeplitz systems $A_n x = b$, fast and superfast direct solvers of complexity $O(n^2)$ and $O(n \log^2 n)$, respectively, have been developed; see, for instance, Trench [10] and Ammar and Gragg [1]. However, there exists no fast direct solvers for solving Toeplitz-plus-band systems. This is mainly because the displacement rank of the matrix $A_n + B_n$ can take any value between 0 and n. Hence fast Toeplitz solvers that are based on small displacement rank of matrices cannot be applied.

We note that given any vector x, the product $(A_n + B_n)x$ can be computed in $O(n \log n)$ operations. In fact, $A_n x$ can be obtained by fast Fourier transform by first embedding A_n into a $2n$-by-$2n$ circulant matrix; see Strang [9]. Thus iterative methods such as the conjugate gradient method can be employed for solving these systems. The convergence rate of the conjugate gradient method depends on the spectrum of the matrix $A_n + B_n$; see Golub and Van Loan [8]. However, generally, the spectrum of A_n, and hence of $A_n + B_n$, is not clustered and the method will therefore converge slowly. Hence a suitable preconditioner should be chosen to speed up the convergence.

For Toeplitz systems $A_n x = b$, circulant preconditioners have been proved to be successful choices under the assumption that the diagonals of A_n are Fourier coefficients of a positive 2π-periodic continuous function. In that case, Chan and Yeung [3] proved that

*Received by the editors November 18, 1991; accepted for publication (in revised form) August 17, 1992.
†Department of Mathematics, University of Hong Kong, Hong Kong.

the convergence rate of the method is superlinear. However, circulant preconditioners do not work for Toeplitz-plus-band systems. In fact, Strang's circulant preconditioner [9] is not even defined for non-Toeplitz matrices. T. Chan's circulant preconditioner, while defined for $A_n + B_n$, will not work well when the eigenvalues of B_n are not clustered; see the numerical results in §4. Even if we approximate A_n by a circulant preconditioner M_n, the matrix $M_n + B_n$ cannot be used as a preconditioner since the system $(M_n + B_n)z = y$ cannot be solved easily.

In this paper, we use band matrices C_n as preconditioners. We will assume that B_n is an arbitrary Hermitian positive semidefinite band matrix with bandwidth independent of n, and the diagonals of A_n are Fourier coefficients of a nonnegative piecewise continuous function f. In that case, $A_n + B_n$ will be Hermitian positive definite. We prove that if the essential infimum of f is attained by finitely many points in $[-\pi, \pi]$ and if f is sufficiently smooth around these points, then there exists a Hermitian positive definite band matrix C_n, with bandwidth independent of n, such that the spectra of $C_n^{-1}(A_n + B_n)$ are uniformly bounded by a constant independent of n. Hence for a given tolerance, the number of iterations required for convergence is independent of n. Since the band matrix system $C_n x = b$ can be solved in $O(n)$ operations, the total complexity of the method is $O(n \log n)$.

The outline of the rest of the paper is as follows. In §2, we introduce our preconditioners C_n and study the spectral properties of $A_n + B_n$ and C_n. In §3, we show that the spectra of $C_n^{-1}(A_n + B_n)$ are uniformly bounded by constants independent of n. Finally, numerical examples and concluding remarks are given in §4.

2. Construction of the preconditioner C_n.

To begin with, let $\mathcal{C}^+$ be the set of all nonnegative piecewise continuous functions defined on $[-\pi, \pi]$. For all $f \in \mathcal{C}^+$, let

$$t_k[f] = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\theta) e^{-ik\theta} d\theta, \qquad k = 0, \pm 1, \pm 2, \ldots$$

be the Fourier coefficients of f. Since f is real-valued,

$$t_{-k}[f] = \bar{t}_k[f], \qquad k = 0, \pm 1, \pm 2, \ldots.$$

Let $A_n[f]$ be the n-by-n Hermitian Toeplitz matrix with the (j, l)th entry given by $t_{j-l}[f]$. The function f is called the generating function of the matrices $A_n[f]$. We recall that a point θ_0 is said to be a zero of f with order ν if $f(\theta_0) = 0$ and ν is the smallest positive integer such that $f^{(\nu)}(\theta_0) \neq 0$ and $f^{(\nu+1)}(\theta)$ is continuous in a neighborhood of θ_0.

In the following, we denote the essential infimum and the essential supremum of f by $f_{\min}$ and $f_{\max}$, respectively. We will assume that f attains $f_{\min}$ at finitely many points in $[-\pi, \pi]$ and that f is smooth around these points. More precisely, we assume that $f(\theta) - f_{\min}$ has finitely many zeros in $[-\pi, \pi]$ and that the orders of these zeros are finite and positive. Notice that the matrix $A_n[f]$ is unchanged when f is redefined at finitely many points. Thus we can always assume without loss of generality that f is continuous at those minimum points.

From the assumptions, we see that $f_{\max} \neq f_{\min}$. Then by using the fact that

$$u^* A_n[g] u = \frac{1}{2\pi} \int_{-\pi}^{\pi} |\sum_{j=1}^{n} u_j e^{i(j-1)\theta}|^2 g(\theta) d\theta \tag{1}$$

for any $g \in \mathcal{C}^+$ and any n-vector $u = (u_1, \ldots, u_n)^*$, Chan [4, Lemma 1] proved that

$$\lambda_{\min}(A_n[f]) > f_{\min}. \tag{2}$$

Here $\lambda_{\min}(A_n[f])$ is the smallest eigenvalue of $A_n[f]$. Since f is nonnegative, $A_n[f]$ is positive definite for all n. In the following, we will assume that the band matrices B_n are Hermitian positive semidefinite matrices with bandwidth $2\omega + 1$ and that ω is independent of n. Clearly the matrix $A_n[f] + B_n$ is positive definite for all n.

For all $n > 0$, our preconditioners C_n are defined as

$$C_n \equiv A_n[b_\mu] + B_n + f_{\min} \cdot I_n. \tag{3}$$

Here

$$b_\mu(\theta) = (2 - 2\cos\theta)^\mu = \left(2\sin\left(\frac{\theta}{2}\right)\right)^{2\mu},$$

and it has a unique zero of order 2μ at $\theta = 0$. We remark that $A_n[b_\mu]$ is a symmetric band Toeplitz matrix of bandwidth $(2\mu + 1)$ and its diagonals are given by the Pascal triangle; see Chan [4]. Clearly, C_n is a symmetric band matrix of bandwidth

$$2\ell + 1 = \max\{2\mu + 1, 2\omega + 1\}.$$

Moreover, since the minimum of b_μ is 0, it follows from (2) and (3) that

$$\lambda_{\min}(C_n) \geq \lambda_{\min}(A_n[b_\mu]) + \lambda_{\min}(B_n) + f_{\min} > f_{\min} \geq 0.$$

In particular, the preconditioner C_n is positive definite for all n. We note that in [4, Thm. 2], we have shown that $A_n[b_\mu(\theta)] + f_{\min} \cdot I_n$ is a good preconditioner for $A_n[f]$. Thus intuitively, we expect C_n so defined to be a good preconditioner for $A_n[f] + B_n$.

3. Condition number of the preconditioned matrix. In this section, we show that the spectra of $C_n^{-1}(A_n + B_n)$ are uniformly bounded by constants independent of n. We first consider generating functions f in C^+ where $f(\theta) - f_{\min}$ has only one zero at θ_0. Let the order of θ_0 be ν. We note that $f^{(\nu)}(\theta_0) > 0$ and ν must be even. We remark also that we can assume without loss of generality that $\theta_0 = 0$. In fact the function $f(\theta + \theta_0) - f_{\min}$ has a zero at $\theta = 0$ and

$$A_n[f(\theta + \theta_0)] = V_n^* A_n[f(\theta)] V_n,$$

where $V_n = \operatorname{diag}(1, e^{-i\theta_0}, e^{-2i\theta_0}, \ldots, e^{-i(n-1)\theta_0})$; see Chan [4, Lemma 2].

THEOREM 1. *Let $f \in C^+$. Suppose that $f(\theta) - f_{\min}$ has a unique zero at $\theta = 0$ with order equal to 2μ. Let $C_n = A_n[b_\mu] + B_n + f_{\min} \cdot I$. Then $\kappa(C_n^{-1}(A_n[f] + B_n))$ is uniformly bounded for all $n > 0$.*

Proof. By assumption, there exists a neighborhood N of 0 such that f is continuous in N. Define

$$F(\theta) = \frac{f(\theta)}{(2 - 2\cos\theta)^\mu + f_{\min}}.$$

Clearly F is continuous and positive for $\theta \in N \setminus \{0\}$. Since

$$\lim_{\theta \to 0} F(\theta) = \begin{cases} 1 & f_{\min} > 0, \\ \dfrac{f^{(2\mu)}(0)}{(2\mu)!} & f_{\min} = 0 \end{cases}$$

is positive, F is a continuous positive function in N. Since f is piecewise continuous and positive almost everywhere in $[-\pi, \pi] \setminus N$, we see that F is a piecewise continuous function with a positive essential infimum in $[-\pi, \pi]$. Hence there exist constants $b_1, b_2 > 0$, such that $b_1 \le F(\theta) \le b_2$ almost everywhere in $[-\pi, \pi]$. Without loss of generality, we assume that $b_2 \ge 1 \ge b_1$. By using (1), we then have

$$b_1 \le \frac{u^* A_n[f] u}{u^*(A_n[b_\mu] + f_{\min} \cdot I_n) u} \le b_2$$

for any n-vector u. Recall that B_n is positive semidefinite and $C_n = A_n[b_\mu] + B_n + f_{\min} \cdot I_n$. We then have

$$b_1 \le \frac{u^*(A_n[f] + B_n) u}{u^* C_n u} \le b_2$$

for any n-vector u. Hence $\kappa(C_n^{-1}(A_n[f] + B_n)) \le b_2/b_1$, which is independent of n. □

We remark that the results can be readily generalized to the case where $f_{\min}$ is attained at finitely many points; cf. Chan [4, Thm. 3]. The bandwidth of C_n will be given by

$$2\ell + 1 = \max\left\{ \sum_j \nu_j + 1, 2\omega + 1 \right\},$$

where ν_j are the orders of the zeros of $f(\theta) - f_{\min}$ and the summation is over all such zeros.

Next we consider the computational cost and storage requirement of our method. The number of operations per iteration in the preconditioned conjugate gradient method depends mainly on the work of computing the matrix–vector multiplication $C_n^{-1}(A_n[f] + B_n)y$; see, for instance, Golub and Van Loan [8]. In this case, the matrix–vector multiplication $B_n y$ requires only $(2\omega + 1)n$ operations and the product $A_n[f]y$ can be done in $O(n \log n)$ operations by the fast Fourier transform. The system $C_n y = z$ can be solved by using any band matrix solver. The cost of factorizing C_n is about $\frac{1}{2}\ell^2 n$ operations and then each subsequent solve requires an extra $(2\ell + 1)n$ operations. Hence the total operations per iteration is of order $O(n \log n)$ as ℓ and ω are independent of n. It is well known that the number of iterations required to attain a given tolerance ϵ is bounded by

$$\frac{1}{2}\sqrt{\kappa(C_n^{-1}(A_n + B_n))} \log\left(\frac{1}{\epsilon}\right) + 1.$$

Since the condition number is uniformly bounded in this case, the overall work required to attain the given tolerance is of $O(n \log n)$ operations.

As for the storage, we need five n-vectors in the conjugate gradient method. The diagonals of A_n and the bands of B_n require extra $(\omega + 2)$ n-vectors, and finally, we need an n-by-$(\ell + 1)$ matrix to hold the factors of the preconditoner C_n. Thus the overall storage requirement is about $(8 + \ell + \omega)n$, which is significantly less than the $O(n^2)$ storage required by Gaussian elimination method.

4. Numerical results and concluding remarks.

To test the convergence rate of the preconditioner, we considered two different band matrices. The first one is the diagonal matrix

$$D_n = f_{\max} \cdot \operatorname{diag}\left[0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}\right]$$

TABLE 1
Number of iterations for $B_n = D_n$.

f	θ^4				$\cosh\theta$				$J(\theta)$			
n	No	C_n	E_n	T_n	No	C_n	E_n	T_n	No	C_n	E_n	T_n
16	16	9	8	16	15	8	8	15	14	12	9	14
32	26	11	9	23	21	9	9	18	18	14	12	16
64	36	12	11	31	25	9	10	21	23	14	15	19
128	50	14	16	40	29	10	11	23	30	15	18	24
256	68	15	21	53	32	10	12	25	39	15	23	30
512	91	15	32	70	34	10	14	27	50	15	28	38
1024	122	16	64	91	36	10	16	27	63	15	35	47

whose eigenvalues are distributed uniformly in the interval $[0, f_{\max}]$. The second one is a symmetric tridiagonal matrix given by

$$B_n^{(\alpha)} = (n+1)^\alpha \cdot \frac{2\pi}{n+1} \begin{bmatrix} 2 & -\frac{3}{2} & & & \\ -\frac{3}{2} & 4 & -\frac{5}{2} & & \\ & -\frac{5}{2} & 6 & \ddots & \\ & & \ddots & \ddots & -\frac{2n-1}{2} \\ & & & -\frac{2n-1}{2} & 2n \end{bmatrix}.$$

Notice that $B_n^{(2)}$ is the discretization matrix of the operator

$$\frac{d}{d\theta}\left\{(\theta+\pi)\frac{d}{d\theta}\right\}$$

in $[-\pi, \pi]$. Clearly, the matrices $B_n^{(\alpha)}$ are irreducibly diagonally dominant; hence they are positive definite.

In our tests, the vector of all ones is the right-hand side vector, the zero vector is the initial guess, and the stopping citerion is $||r_q||_2/||r_0||_2 \le 10^{-7}$, where r_q is the residual vector after q iterations. The computations are done with 8-byte arithmetic on a Vax 6420. Three different generating functions were tested. They are θ^4, $\cosh\theta$, and

$$J(\theta) \equiv \begin{cases} \theta^2 & |\theta| \le \pi/2, \\ 1 & |\theta| > \pi/2. \end{cases}$$

The corresponding bandwidths of C_n are 5, 3, and 3, respectively.

For comparison, we also solved the problems with two other preconditioners. The first one is the T. Chan circulant preconditioner T_n corresponding to the matrix $A_n[f] + B_n$. The second preconditioner E_n, which has the same bandwidth as C_n, is obtained by just copying the diagonals of $A_n[f]+B_n$. We note that some of the E_n may be indefinite. In contrast, C_n and T_n are always positive definite. Tables 1–4 show the number of iterations required for convergence (** means more than 1000 iterations). We see that as n increases, the number of iterations stays almost the same when C_n is used as the preconditioner while it increases if others are used.

We conclude that our algorithm solves the system $(A_n + B_n)x = b$ in $O(n \log n)$ operations for a certain class of Toeplitz matrices A_n. The cost is significantly less than the $O(n^3)$ cost required by Gaussian elimination method. We note that the spectra of $C_n^{-1}(A_n[f] + B_n)$ generally will not be clustered around 1 although they are uniformly

TABLE 2
Number of iterations for $B_n = B_n^{(0)}$.

f	θ^4				$\cosh\theta$				$J(\theta)$			
n	No	C_n	E_n	T_n	No	C_n	E_n	T_n	No	C_n	E_n	T_n
16	17	12	15	16	16	7	9	14	16	9	9	16
32	42	15	35	23	31	8	13	16	35	10	14	27
64	107	17	98	32	38	9	18	17	75	12	23	35
128	268	19	372	45	42	9	30	17	162	14	40	42
256	652	21	**	63	43	9	48	17	329	16	75	51
512	**	22	**	90	43	10	82	17	670	17	146	61
1024	**	23	**	127	43	10	146	16	**	18	293	74

TABLE 3
Number of iterations for $B_n = B_n^{(1)}$.

f	θ^4				$\cosh\theta$				$J(\theta)$			
n	No	C_n	E_n	T_n	No	C_n	E_n	T_n	No	C_n	E_n	T_n
16	16	8	8	16	16	5	6	16	16	5	5	16
32	37	8	10	31	36	5	7	31	36	5	6	32
64	82	8	13	47	82	5	8	46	83	5	8	51
128	188	8	18	70	184	5	9	65	189	5	9	77
256	415	8	23	104	408	5	12	91	418	5	11	112
512	893	8	31	152	826	5	13	130	896	5	14	164
1024	**	8	40	220	**	5	16	183	**	5	17	238

TABLE 4
Number of iterations for $B_n = B_n^{(2)}$.

f	θ^4				$\cosh\theta$				$J(\theta)$			
n	No	C_n	E_n	T_n	No	C_n	E_n	T_n	No	C_n	E_n	T_n
16	16	4	5	17	16	3	4	16	16	3	3	16
32	37	4	5	32	37	3	4	32	37	3	4	32
64	83	4	5	52	83	3	4	52	83	3	4	52
128	189	3	5	77	190	3	4	77	190	3	4	77
256	418	3	5	113	417	3	4	114	418	3	4	113
512	897	3	5	166	897	2	4	165	898	2	4	166
1024	**	3	5	241	**	2	4	240	**	2	4	241

bounded. We finally remark that our results in this paper extend those obtained in Chan [4]. More precisely, in [4], we proved that $\kappa(C_n^{-1}A_n[f])$ is uniformly bounded whenever f is 2π-periodic continuous. However, using Theorem 1 with B_n equal to a zero matrix, we see that the same conclusion holds whenever f is 2π-periodic piecewise continuous.

Acknowledgment. We would like to thank Professor Israel Koltracht of the University of Connecticut for his valuable suggestions.

REFERENCES

[1] G. AMMAR AND W. GRAGG, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Appl., 9 (1988), pp. 61–76.

[2] G. CARAYANNIS, N. KALOUPTSIDIS, AND D. MANOLAKIS, *Fast recursive algorithms for a class of linear equations*, IEEE Trans. Acoust. Speech Signal Process., 30 (1982), pp. 227–239.

[3] R. CHAN AND M. YEUNG, *Circulant preconditioners for Toeplitz matrices with positive continuous generating functions*, Math. Comp., to appear.

[4] R. CHAN, *Toeplitz preconditioners for Toeplitz systems with nonnegative generating functions*, IMA J. Numer. Anal., 11 (1991), pp. 333–345.

[5] T. CHAN, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 766–771.
[6] L. DELVES AND J. MOHAMED, *Computational Methods for Integral Equations,* Cambridge University Press, Cambridge, England, 1985.
[7] U. GRENANDER AND G. SZEGÖ, *Toeplitz Forms and their Applications*, 2nd ed., Chelsea Publishing Co., New York, 1984.
[8] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.
[9] G. STRANG, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.
[10] W. TRENCH, *An algorithm for the inversion of finite Toeplitz matrices*, SIAM J. Appl. Math., 12 (1964), pp. 515–522.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1020–1033, September 1993

VARIANTS OF BICGSTAB FOR MATRICES WITH COMPLEX SPECTRUM*

MARTIN H. GUTKNECHT†

Abstract. Recently Van der Vorst [*SIAM J. Sci. Statist. Comput.*, 13 (1992), pp. 631–644] proposed for solving nonsymmetric linear systems $Az = b$ a biconjugate gradient (BICG)-based Krylov space method called BICGSTAB that, like the biconjugate gradient squared (BICGS) method of Sonneveld, does not require matrix-vector multiplications with the transposed matrix A^T, and that has typically a much smoother convergence behavior than BICG and BICGS. Its nth residual polynomial is the product of the one of BICG (i.e., the nth Lanczos polynomial) with a polynomial of the same degree with real zeros. Therefore, nonreal eigenvalues of A are not approximated well by the second polynomial factor. Here, the author presents for real nonsymmetric matrices a method BICGSTAB2 in which the second factor may have complex conjugate zeros. Moreover, versions suitable for complex matrices are given for both methods.

Key words. Lanczos algorithm, biconjugate gradient algorithm, conjugate gradient squared algorithm, BICGSTAB, formal orthogonal polynomial, nonsymmetric linear system, Krylov space method

AMS subject classification. 65F10

1. From BICG to complex BICGSTAB.

The *biconjugate gradient method* (BICG) of Lanczos [7] and Fletcher [1] is a Krylov space method for solving (real or complex) non-Hermitian linear system $Az = b$, where A is, say, a nonsingular $N \times N$ matrix. (Typically, this matrix will be the result of applying a preconditioner to the original system matrix.) Starting from some initial guess z_0 for the solution, BICG generates a sequence z_n with the property that the nth residual $r_n := b - Az_n$ lies in the Krylov space generated by A from r_0, i.e.,

$$r_n \in \mathcal{K}_{n+1} := \text{span}\,(r_0, Ar_0, \ldots, A^n r_0), \tag{1}$$

and is orthogonal to another Krylov space generated from some other initial vector y_0 by the Hermitian transpose A^H

$$r_n \perp \mathcal{L}_n := \text{span}\,(y_0, A^H y_0, \ldots, (A^H)^{n-1} y_0). \tag{2}$$

The sequence of residual polynomials ρ_n, which are implicitly defined by

$$r_n = \rho_n(A) r_0, \tag{3}$$

is in view of (2) a sequence of formal orthogonal polynomials: if we define a linear functional Φ on the space of polynomials with complex coefficients by setting $\Phi(\zeta^k) := y_0^H A^k x_0$, the formal orthogonality relation $\Phi(\pi_k \rho_n) = 0$ holds for every polynomial π_k of degree $k < n$; see [5], [3] for further details and references. As a consequence of the consistency condition for polynomial acceleration methods, these residual polynomials are normalized by $\rho_n(0) = 1$. They are often called *Lanczos polynomials*. In general, neither these polynomials nor the residuals satisfy a minimality condition, in contrast to the case in which A is Hermitian positive definite and $y_0 = r_0$, where the method reduces to the classical conjugate gradient method. Theoretically, the BICG algorithm terminates in at most $\nu(A, r_0)$ steps if this number denotes the degree of the minimal polynomial of the restriction of A to the maximum Krylov space generated by A from

*Received by the editors September 9, 1991; accepted for publication (in revised form) August 17, 1992.

†Interdisciplinary Project Center for Supercomputing (IPS), ETH Zürich, ETH-Zentrum, CH-8092 Zürich, Switzerland (mhg@ips.ethz.ch).

r_0. In practice this property is often irrelevant because $\nu(A, r_0)$ may be very large and the orthogonality (2) is partly lost due to roundoff. What counts in practice is the good convergence behavior and the small memory requirement of the method.

The algorithm can break down, namely, due to

$$r_n \perp \mathcal{L}_{n+1}; \tag{4}$$

but in most cases such a breakdown or a corresponding near-breakdown can be overcome by a look-ahead step, see [8], [3], [4], [2] and further references cited there.

In the standard version of the BiCG method, which we call BioMin, a second sequence $\{\sigma_n\}$ of formal orthogonal polynomials plays a role. The algorithm is based on the following mixed recurrence formulas (in which we suppress the independent variable ζ by writing ρ_n instead of $\rho_n(\zeta)$, and σ_n instead of $\sigma_n(\zeta)$):

$$\rho_{n+1} := \rho_n - \omega_n \zeta \sigma_n, \tag{5a}$$

$$\sigma_{n+1} := \rho_{n+1} - \psi_{n+1}\sigma_n, \tag{5b}$$

where the coefficients ω_n and ψ_{n+1} are computed as indicated below. These formulas are implicitly used to generate two pairs of finite vector sequences

$$x_n := \rho_n(A)x_0, \quad y_n := \overline{\rho_n}(A^H)y_0, \tag{6a}$$

$$u_n := \sigma_n(A)x_0, \quad v_n := \overline{\sigma_n}(A^H)y_0, \tag{6b}$$

satisfying the biorthogonality conditions

$$y_n^H x_m = \delta_n \delta_{m,n}, \tag{7a}$$

$$v_n^H A u_m = \delta'_n \delta_{m,n}. \tag{7b}$$

(In (6), $\overline{\rho_n}$ and $\overline{\sigma_n}$ have the complex conjugate coefficients of ρ_n and σ_n, and in (7), $\delta_n \neq 0$, $\delta'_n \neq 0$ are constants and $\delta_{m,n}$ is the Kronecker symbol. The scaling of the vectors y_n and v_n is, theoretically, irrelevant; but in practice, a scaling different from the one chosen here may be more appropriate, e.g., one may choose y_n and v_n to have unit length.)

Theoretically one proceeds until the algorithm terminates or breaks down; in practice, one stops when the residual is sufficiently small. Here, $x_n = r_n$ is actually the nth residual vector of BiCG, but as we will see, there are other algorithms that are also based on the recursions (5) yet have other residual vectors. In any case, the recurrences (5) are not used to generate the polynomials, but are translated into recurrences for the vectors specified by (6). For the iterates z_n, an additional recurrence related to the one for the residual vectors $x_n = r_n$ is easily derived. The coefficients ω_n and ψ_{n+1} in (5) are determined from the biorthogonality conditions (7) and the consistency condition $\rho_n(0) = 1$ (see [5]); in fact, only the case $m = n + 1$ of (7) is enforced:

$$\omega_n := \delta_n / (y_n^H A u_n) = \delta_n / (v_n^H A u_n), \tag{8a}$$

$$\delta_{n+1} := y_{n+1}^H x_{n+1}, \tag{8b}$$

$$\psi_{n+1} := (y_{n+1}^H A u_n)/(v_n^H A u_n) = -\delta_{n+1}/\delta_n. \tag{8c}$$

(In the equalities in (8a) and (8c), one makes use of the formal orthogonality of the polynomials and of the fact that, in view of (5), ρ_{n+1}, σ_{n+1}, and $-\omega_n \zeta \sigma_n$ have the same leading coefficient.)

It is the great advantage of the Lanczos approach based on (5) that observing the conditions (7) for $m = n+1$ implies theoretically that they hold for all $m \le n$. The short recurrences (5) mean very small memory requirements. Moreover, although in practice the biorthogonality with respect to the earlier constructed vectors (i.e., for large $|m-n|$) tends to get lost due to roundoff, it is this biorthogonality that accounts for the normally observed gradual decrease of the residual length. The biorthogonality is related to a Padé approximation problem. However, the convergence is still not well understood, and in practice is often far from monotonic. But for very large problems, Lanczos-type methods, due to their small memory requirements, are often the best choice.

One disadvantage of the unsymmetric Lanczos process and BICG is that the construction of y_n requires one application of A^H in each step in addition to the one application of A to compute z_n and r_n. (In BICG, u_n and v_n are then found without further matrix–vector products.) In 1984 Sonneveld [11] with his *(bi)conjugate gradient squared* (CGS or BICGS) *method* proposed a way to get around A^H. He uses a set of recurrences that allows him to compute other iterates z_n whose residuals $r_n := b - Az_n$ satisfy

$$r_n = \rho_n^2(A)r_0, \tag{9}$$

where the polynomials ρ_n are still the Lanczos polynomials. In fact, the iterates and their residuals, together with the vectors defined by

$$p_n := \rho_n(A)\sigma_n(A)r_0, \quad q_n := \rho_n(A)\sigma_{n-1}(A)r_0, \quad s_n := \sigma_n^2(A)r_0, \tag{10}$$

can be generated by the Krylov space analogs of the polynomial recurrences

$$\begin{aligned}
\rho_{n+1}\sigma_n &= \rho_n\sigma_n - \omega_n\zeta\sigma_n^2, && \text{(11a)}\\
\rho_{n+1}\sigma_{n+1} &= \rho_{n+1}^2 - \psi_{n+1}\rho_{n+1}\sigma_n, && \text{(11b)}\\
\rho_{n+1}^2 &= \rho_n^2 - \omega_n\zeta(\rho_n\sigma_n + \rho_{n+1}\sigma_n), && \text{(11c)}\\
\sigma_{n+1}^2 &= \rho_{n+1}\sigma_{n+1} - \psi_{n+1}\rho_{n+1}\sigma_n + \psi_{n+1}^2\sigma_n^2, && \text{(11d)}
\end{aligned}$$

which are readily derived from (5). Due to the squaring of the residual polynomial, the sometimes erratic convergence behavior of BICG is even more pronounced here.

Of course, the recurrences (11) need to be complemented by formulas for determining ω_n and ψ_{n+1}. Using (6) and (10), the expressions (8) are readily transformed into

$$\begin{aligned}
\omega_n &= \delta_n/(y_0^H A s_n), && \text{(12a)}\\
\delta_{n+1} &= y_0^H r_{n+1}, && \text{(12b)}\\
\psi_{n+1} &= -\delta_{n+1}/\delta_n. && \text{(12c)}
\end{aligned}$$

As mentioned before, these coefficients ω_n and ψ_{n+1} result basically from observing the biorthogonality conditions (7) for $m = n+1$, i.e.,

$$y_n^H x_{n+1} = 0, \qquad v_n^H A u_{n+1} = 0. \tag{13}$$

At this point it is worth noting that for $m \ne n$, (7) is equivalent to

$$\begin{aligned}
&y_n \perp \mathcal{K}_n, \;\; x_n \perp \mathcal{L}_n\,, && \text{(14a)}\\
&A^H v_n \perp \mathcal{K}_n, \;\; A u_n \perp \mathcal{L}_n\,. && \text{(14b)}
\end{aligned}$$

In view of (6) these conditions are not independent; those on the left-hand side are equivalent to those on the right-hand side. If $\mathcal{P}_n$ denotes the set of polynomials of degree at most n, the latter can be expressed as

$$y_0^H \pi_n(A) x_{n+1} = 0 \quad (\forall \pi_n \in \mathcal{P}_n), \tag{15a}$$
$$v_0^H \tilde{\pi}_n(A) A u_{n+1} = 0 \quad (\forall \tilde{\pi}_n \in \mathcal{P}_n). \tag{15b}$$

The conditions (13) are just the special case of (15) with $\pi_n = \rho_n$ and $\tilde{\pi}_n = \sigma_n$. But the same restrictions can be taken into account by choosing for π_n and $\tilde{\pi}_n$ any other polynomial of exact degree n. Van der Vorst [14] (following an earlier proposal by Sonneveld) discovered that a simple implementation and an often excellent convergence behavior can be attained by choosing for π_n and $\tilde{\pi}_n$ polynomials τ_n that are built up in factored form

$$\tau_n(\zeta) = (1 - \chi_0 \zeta)(1 - \chi_1 \zeta) \cdots (1 - \chi_{n-1} \zeta) \tag{16}$$

by adding a suitable new zero $1/\chi_n$ at each step. Clearly,

$$\tau_{n+1} = (1 - \chi_n \zeta) \tau_n, \tag{17}$$

so that

$$\rho_{n+1} \tau_{n+1} = (1 - \chi_n \zeta) \rho_{n+1} \tau_n. \tag{18}$$

Multiplication of (5a) and (5b) by τ_n and τ_{n+1} yields the further recurrences

$$\rho_{n+1} \tau_n = \rho_n \tau_n - \omega_n \zeta \sigma_n \tau_n, \tag{19}$$
$$\sigma_{n+1} \tau_{n+1} = \rho_{n+1} \tau_{n+1} - \psi_{n+1} (1 - \chi_n \zeta) \sigma_n \tau_n. \tag{20}$$

Equations (18)–(20) are a set of recurrence relations that after the translation from polynomials to Krylov space vectors can be used to build up the vector sequences

$$\tilde{r}_n := \rho_n(A) \tau_n(A) r_0, \quad \tilde{w}_n := \rho_n(A) \tau_{n-1}(A) r_0, \quad \tilde{s}_n := \sigma_n(A) \tau_n(A) r_0. \tag{21}$$

Van der Vorst [14], assuming real data (i.e., a real matrix A and $b, z_0, y_0 \in \mathbb{R}^N$), chooses in the nth step of BiCGStab the new zero $1/\chi_n$ so that the new residual norm $||\tilde{r}_{n+1}|| = ||\tilde{w}_{n+1} - A\tilde{w}_{n+1}\chi_n||$ is minimized over $\chi_n \in \mathbb{R}$. Here and in the following, the norm is the Euclidean one. Our first aim is to generalize this approach to complex data. In particular, we allow $\chi_n \in \mathbb{C}$ now. Starting from

$$||\tilde{r}_{n+1}||^2 = ||\tilde{w}_{n+1} - A\tilde{w}_{n+1}\chi_n||^2 = ||\tilde{w}_{n+1}||^2 - 2\,\mathrm{Re}\,\{\tilde{w}_{n+1}^H A \tilde{w}_{n+1} \chi_n\} + \chi_n^2\, ||A\tilde{w}_{n+1}||^2, \tag{22}$$

one can conclude by inserting $\chi_n = \xi_n + i\eta_n$, and computing the partial derivatives $\partial/\partial\xi_n$ and $\partial/\partial\eta_n$, that this minimization leads to

$$\chi_n := \frac{(A\tilde{w}_{n+1})^H \tilde{w}_{n+1}}{||A\tilde{w}_{n+1}||^2}. \tag{23}$$

This is no surprise, since one obtains the same formula easier by noting that the minimization problem can be solved by orthogonal projection of $\tilde{w}_{n+1}$ onto $A\tilde{w}_{n+1}$ in $\mathbb{C}^N$; the image of this projection must be equal to $A\tilde{w}_{n+1}\chi_n$. (If we had restricted χ_n to be real, we would have obtained $\chi_n := \mathrm{Re}\,\{(A\tilde{w}_{n+1})^H \tilde{w}_{n+1}\}/||A\tilde{w}_{n+1}||^2$ instead.)

Formulas for the coefficients ω_n and ψ_{n+1} are found by modifying (12) for taking the leading coefficient $(-1)^n\chi_0 \cdots \chi_{n-1}$ of τ_n into account. Noting that σ_n and ρ_n have leading coefficient $(-1)^n\omega_0 \cdots \omega_{n-1}$ and setting $\tilde{\delta}_n := \delta_n(\chi_0 \cdots \chi_{n-1})/(\omega_0 \cdots \omega_{n-1})$, we get

$$\text{(24a)} \qquad \omega_n = \frac{\delta_n}{y_0^H A s_n} = \frac{\delta_n}{y_0^H A \sigma_n^2(A) x_0} = \frac{\tilde{\delta}_n}{y_0^H A \sigma_n(A)\tau_n(A) x_0} = \frac{\tilde{\delta}_n}{y_0^H A \tilde{s}_n},$$

$$\text{(24b)} \qquad \tilde{\delta}_{n+1} = y_0^H \tilde{r}_{n+1},$$

$$\text{(24c)} \qquad \psi_{n+1} = -\frac{\delta_{n+1}}{\delta_n} = \frac{\tilde{\delta}_{n+1}\omega_n}{\tilde{\delta}_n \chi_n}.$$

Altogether, one obtains the following complex version of BiCGStab. If applied to real data, it is identical with Van der Vorst's algorithm. The letter o denotes the zero vector.

ALGORITHM 1 (BiCGStab). *For solving $Az = b$ choose an initial approximation $z_0 \in \mathbb{C}^N$ and set $\tilde{r}_0 := \tilde{s}_0 := b - Az_0$. Choose $y_0 \in \mathbb{C}^N$ such that $\tilde{\delta}_0 := y_0^H \tilde{r}_0 \neq 0$ and $\varphi_0 := y_0^H A \tilde{s}_0 / \tilde{\delta}_0 \neq 0$. Then compute for $n = 0, 1, \ldots$*

$$\text{(25a)} \qquad \omega_n := 1/\varphi_n,$$

$$\text{(25b)} \qquad \tilde{w}_{n+1} := \tilde{r}_n - A\tilde{s}_n\omega_n,$$

$$\text{(25c)} \qquad \chi_n := (A\tilde{w}_{n+1})^H \tilde{w}_{n+1} / \|A\tilde{w}_{n+1}\|^2,$$

$$\text{(25d)} \qquad \tilde{r}_{n+1} := \tilde{w}_{n+1} - A\tilde{w}_{n+1}\chi_n,$$

$$\text{(25e)} \qquad z_{n+1} := z_n + \tilde{s}_n\omega_n + \tilde{w}_{n+1}\chi_n,$$

$$\text{(25f)} \qquad \tilde{\delta}_{n+1} := y_0^H \tilde{r}_{n+1},$$

$$\text{(25g)} \qquad \psi_{n+1} := -\omega_n\tilde{\delta}_{n+1}/(\tilde{\delta}_n\chi_n),$$

$$\text{(25h)} \qquad \tilde{s}_{n+1} := \tilde{r}_{n+1} - (\tilde{s}_n - A\tilde{s}_n\chi_n)\psi_{n+1},$$

$$\text{(25i)} \qquad \varphi_{n+1} := y_0^H A \tilde{s}_{n+1} / \tilde{\delta}_{n+1}.$$

If $\tilde{r}_{n+1} = o$, the process terminates and z_{n+1} is the solution of $Az = b$; if $\tilde{r}_{n+1} \neq o$ but $\tilde{\delta}_{n+1} = 0$ or $\varphi_{n+1} = 0$, the algorithm breaks down.

We need to comment on the conditions under which this algorithm breaks down. Clearly, in exact arithmetic, a breakdown of BiCG, caused by $\varphi_n = 0$ or $\delta_n = 0$, see [5], is paralleled by a breakdown of BiCGStab, caused by $\varphi_n = 0$ or $\tilde{\delta}_n = 0$, since $\tilde{\delta}_n := \delta_n(\chi_0 \cdots \chi_{n-1})/(\omega_0 \cdots \omega_{n-1})$. However, in the above formulation, BiCGStab obviously also breaks down if $\chi_n = 0$. A closer look shows that ψ_{n+1} may still be finite in this case, cf. (24c), so that one might try to find an alternative formula for (25g). However, if $\chi_n = 0$, then $\tilde{r}_{n+1} = \tilde{w}_{n+1} = \rho_{n+1}(A)\tau_n(A)r_0$, and it follows from (15a) that $\tilde{\delta}_{n+1} := y_0^H \tilde{r}_{n+1} = y_0^H \rho_{n+1}(A)\tau_n(A)r_0 = y_0^H \tau_n(A) x_{n+1} = 0$. Consequently, $\chi_n = 0$ implies $\tilde{\delta}_{n+1} = 0$, so it suffices to check the two conditions $\tilde{\delta}_{n+1} = 0$ and $\varphi_{n+1} = 0$, although this conceals the fact that $\tilde{\delta}_{n+1} = 0$ can be caused by $\delta_{n+1} = 0$ or $\chi_n = 0$. The new pitfall here is that we do not advance in the Krylov space if $\chi_n = 0$. Then $\tilde{r}_{n+1} \in \mathcal{K}_{2n+2}$, but $\tilde{r}_{n+1} = \rho_{n+1}(A)\tau_{n+1}(A)r_0$ should be in $\mathcal{K}_{2n+3}\backslash\mathcal{K}_{2n+2}$.

2. Two-dimensional local minimization: BiCGStab2. As is well known and is indicated by (3), the convergence of a polynomial acceleration method like BiCG hinges on the damping properties of the residual polynomials ρ_n. Ideally, the lemniscates $|\rho_n(\zeta)| = \varepsilon$ should already for some small ε embrace the spectrum of A or, in the case of a nonnormal A, rather the pseudospectrum [12]. The residual polynomials $\rho_n\tau_n$ of BiCGStab combine the damping properties of the Lanczos polynomials ρ_n (which not only depend on A but also on $r_0 = x_0$ and y_0) with those of τ_n. If one had $\rho_n \equiv 1$ for all n (which is impossible since ρ_n has exact degree n), the polynomials τ_n would be the residual polynomials of GMRES(1) [10], i.e., of GMRES restarted at each step. The factor ρ_n causes a modification of the restart vector, but in any case, one cannot expect τ_n to have an excellent global damping effect. Although the success of BICGSTAB shows that τ_n is normally good enough to level out the irregular convergence of the Lanczos polynomials ρ_n, it has an obvious deficiency when the method (in its original version proposed by Van der Vorst) is applied to a real nonsymmetric system: while generally a nonsymmetric real matrix A has a complex spectrum, all the zeros of τ_n are real if x_0 and y_0 are real vectors.[1] It is therefore natural to try to modify the method so that in the real case τ_n may have pairs of complex conjugate zeros. This is the basic idea for the method BICGSTAB2 defined next. We formulate it also for complex data, but the reader must keep in mind that it also brings a major improvement in the real case.

Let us redefine the polynomials τ_n according to $\tau_0 :\equiv 1$ and the recurrences

$$\tau_{2m+1} := (1 - \chi_m\zeta)\tau_{2m}, \tag{26a}$$

$$\tau_{2m+2} := (1 - \xi_m)\tau_{2m} + (\xi_m + \eta_m\zeta)\tau_{2m+1}, \tag{26b}$$

with $\chi_m, \xi_m, \eta_m \in \mathbb{C}$. (In the case of real data, $\chi_m, \xi_m, \eta_m \in \mathbb{R}$.) Note that $\tau_n(0) = 1$ (for all n) by induction. Clearly, τ_{2m+1} has, as in the original BICGSTAB, the new zero $1/\chi_m$, but in the next step τ_{2m+2} is chosen as a linear combination of τ_{2m}, τ_{2m+1} and $\zeta\tau_{2m+1}$, restricted only by $\tau_{2m+2}(0) = 1$. Hence, the zero $1/\chi_m$ is dismissed and τ_{2m} is supplemented by two new zeros, i.e., there holds

$$\tau_{2m+2} = (1 - \zeta_{2m}\zeta)(1 - \zeta_{2m+1}\zeta)\tau_{2m}, \tag{27}$$

where in the case of real data ζ_{2m} and ζ_{2m+1} are either real or complex conjugate. These zeros can, but need not, be computed. The parameters χ_m, ξ_m, and η_m are again chosen to minimize the residual locally (i.e., within a one- or two-dimensional subspace, respectively). But first, we want to look at the recurrences, which are actually independent of these choices.

Clearly, (19) is again valid for all n, and an analogous relation follows by multiplying (5a) with τ_{n-1} instead of τ_n:

$$\rho_{n+1}\tau_n = \rho_n\tau_n - \omega_n\zeta\sigma_n\tau_n, \tag{28}$$

$$\rho_{n+1}\tau_{n-1} = \rho_n\tau_{n-1} - \omega_n\zeta\sigma_n\tau_{n-1} \quad \text{if } n \geq 1. \tag{29}$$

Equations (18) and (20) remain correct for even n after replacing χ_n by χ_m:

$$\rho_{n+1}\tau_{n+1} = (1 - \chi_m\zeta)\rho_{n+1}\tau_n \quad \text{if } n = 2m, \tag{30}$$

$$\sigma_{n+1}\tau_{n+1} = \rho_{n+1}\tau_{n+1} - \psi_{n+1}(1 - \chi_m\zeta)\sigma_n\tau_n \quad \text{if } n = 2m. \tag{31}$$

When n is odd, one obtains instead by using (26b):

$$\rho_{n+1}\tau_{n+1} = (1 - \xi_m)\rho_{n+1}\tau_{n-1} + \xi_m\rho_{n+1}\tau_n + \eta_m\zeta\rho_{n+1}\tau_n \quad \text{if } n = 2m+1. \tag{32}$$

$$\begin{aligned} \sigma_{n+1}\tau_{n+1} &= \rho_{n+1}\tau_{n+1} - \psi_{n+1}\sigma_n\tau_{n+1} \\ &= \rho_{n+1}\tau_{n+1} - \psi_{n+1}[(1 - \xi_m)\sigma_n\tau_{n-1} + (\xi_m + \eta_m\zeta)\sigma_n\tau_n] \quad \text{if } n = 2m+1. \end{aligned} \tag{33}$$

[1]This is analagous to the fact that Newton's method will never find a complex zero of a real polynomial when started on the real axis.

Finally, multiplying (5b) by τ_n, we get for all n

$$\sigma_{n+1}\tau_n = \rho_{n+1}\tau_n - \psi_{n+1}\sigma_n\tau_n. \tag{34}$$

The relations (28)–(34) are a set of recurrence formulas for the products $\rho_n\tau_n$, $\rho_n\tau_{n-1}$, $\rho_n\tau_{n-2}$, $\sigma_n\tau_n$, and $\sigma_n\tau_{n-1}$. Hence, we must build up the vector sequences

$$\text{(35a)}\quad \tilde{r}_n := \rho_n(A)\tau_n(A)r_0, \quad \tilde{w}_n := \rho_n(A)\tau_{n-1}(A)r_0, \quad \approx{w}_n := \rho_n(A)\tau_{n-2}(A)r_0,$$
$$\text{(35b)}\quad \tilde{s}_n := \sigma_n(A)\tau_n(A)r_0, \quad \tilde{t}_n := \sigma_n(A)\tau_{n-1}(A)r_0.$$

The coefficients ω_n and ψ_{n+1} are found by adapting (24). Since (19) and (28) are identical, i.e., since $\tilde{w}_{n+1}$ is given by the same formula (25b) as in BiCGStab, (24a) and (24b) still hold. Because the leading coefficient of τ_n is now $(-1)^m\chi_0\eta_0\cdots\chi_{m-1}\eta_{m-1}$ if $n=2m$ and $(-1)^m\chi_0\eta_0\cdots\chi_{m-1}\eta_{m-1}\chi_m$ if $n=2m+1$, (24c) transforms into

$$\psi_{n+1} = -\frac{\delta_{n+1}}{\delta_n} = \begin{cases} -(\tilde{\delta}_{n+1}\omega_n)/(\tilde{\delta}_n\chi_m) & \text{if } n=2m, \\ +(\tilde{\delta}_{n+1}\omega_n)/(\tilde{\delta}_n\eta_m) & \text{if } n=2m+1. \end{cases} \tag{36}$$

Finally, we need to give formulas for the parameters χ_m, ξ_m, and η_m, which are determined by a one- and a two-dimensional minimization problem, respectively, cf. (26), (30), (32), and (35):

$$\text{(37a)}\quad \|\tilde{r}_{n+1}\| = \|\tilde{w}_{n+1} - A\tilde{w}_{n+1}\chi_m\| = \min \quad \text{if } n=2m,$$
$$\text{(37b)}\quad \|\tilde{r}_{n+1}\| = \|\,\approx{w}_{n+1} + (\tilde{w}_{n+1} - \approx{w}_{n+1})\xi_m + A\tilde{w}_{n+1}\eta_m\| = \min \quad \text{if } n=2m+1.$$

Of course, these minimization problems are again solved by orthogonal projection: as before in (23),

$$\chi_m := \frac{(A\tilde{w}_{n+1})^H\tilde{w}_{n+1}}{\|A\tilde{w}_{n+1}\|^2} \quad \text{if } n=2m. \tag{38}$$

For the projection onto the two-dimensional subspace spanned by $\tilde{w}_{n+1} - \approx{w}_{n+1}$ and $A\tilde{w}_{n+1}$, we define the two-column matrix

$$B_{m+1} := \left[\tilde{w}_{n+1} - \approx{w}_{n+1} \,|\, A\tilde{w}_{n+1}\right] \quad \text{if } n=2m+1, \tag{39a}$$

in terms of which the projection of $\approx{w}_{n+1}$ is given by $B_{m+1}(B_{m+1}^H B_{m+1})^{-1}B_{m+1}^H\,\approx{w}_{n+1}$. Hence, the optimal coefficients ξ_m and η_m for (37b) are

$$\begin{bmatrix}\xi_m \\ \eta_m\end{bmatrix} := -(B_{m+1}^H B_{m+1})^{-1}B_{m+1}^H\,\approx{w}_{n+1} \quad \text{if } n=2m+1. \tag{39b}$$

The 2×2 matrix $B_{m+1}^H B_{m+1}$ can be singular, namely, when the vectors $\tilde{w}_{n+1} - \approx{w}_{n+1}$ and $A\tilde{w}_{n+1}$ are linearly dependent. The solution (ξ_m, η_m) of the minimization problem (37b) is then not unique, and one might want to compute a particular one in a regularized fashion. However, since $\tilde{w}_{n+1} - \approx{w}_{n+1} \in \mathcal{K}_{2n+2}$, this can only happen if $A\tilde{w}_{n+1}$ is in the same space, which again means that we do no longer advance in the Krylov space. Here, assuming no previous breakdown, we can even conclude that $\mathcal{K}_{2n+3} = \mathcal{K}_{2n+2}$ since the polynomial $\zeta\rho_{n+1}(\zeta)\tau_n(\zeta)$ that corresponds to $A\tilde{w}_{n+1}$ has exact degree $2n+2$. In view of $\tilde{r}_{n+1} \in \mathcal{K}_{2n+2}$, we conclude as in the discussion at the end of §1 that $\tilde{\delta}_{n+1} = 0$ due to (15a). The same is true if the minimization problem (37b) has a unique solution, but the second component η_m happens to be zero.

Summarizing we obtain the following two-step algorithm BiCGStab2 for real nonsymmetric or complex linear systems.

Algorithm 2. (BiCGStab2) *For solving $Az = b$ choose an initial approximation $z_0 \in \mathbb{C}^N$ and set $\tilde{r}_0 := \tilde{s}_0 := b - Az_0$. Choose $y_0 \in \mathbb{C}^N$ such that $\delta_0 := y_0^H \tilde{r}_0 \neq 0$ and $\varphi_0 := y_0^H A\tilde{s}_0/\delta_0 \neq 0$. Then compute for $n = 0, 1, \ldots$*

```
(40a)  $\omega_n := 1/\varphi_n$,
(40b)  $\tilde{\tilde{w}}_{n+1} := \tilde{w}_n - A\tilde{t}_n\omega_n$   (if $n \geq 1$),
(40c)  $\tilde{w}_{n+1} := \tilde{r}_n - A\tilde{s}_n\omega_n$;
       if n is even, set $m := n/2$, compute $\chi_m$ by (38) and let
(40d)      $\tilde{r}_{n+1} := \tilde{w}_{n+1} - A\tilde{w}_{n+1}\chi_m$,
(40e)      $z_{n+1} := z_n + \tilde{s}_n\omega_n + \tilde{w}_{n+1}\chi_m$,
(40f)      $\tilde{\delta}_{n+1} := y_0^H\tilde{r}_{n+1}$,
(40g)      $\psi_{n+1} := -\omega_n\tilde{\delta}_{n+1}/(\tilde{\delta}_n\chi_m)$,
(40h)      $\tilde{s}_{n+1} := \tilde{r}_{n+1} - (\tilde{s}_n - A\tilde{s}_n\chi_m)\psi_{n+1}$,
       else set $m := (n-1)/2$, compute $\xi_m$ and $\eta_m$ by (39) and let
(40i)      $\tilde{r}_{n+1} := \tilde{\tilde{w}}_{n+1}(1-\xi_m) + \tilde{w}_{n+1}\xi_m + A\tilde{w}_{n+1}\eta_m$,
(40j)      $z_{n+1} := [z_{n-1} + \tilde{s}_{n-1}\omega_{n-1} + \tilde{t}_n\omega_n](1-\xi_m) + [z_n + \tilde{s}_n\omega_n]\xi_m - \tilde{w}_{n+1}\eta_m$,
(40k)      $\tilde{\delta}_{n+1} := y_0^H\tilde{r}_{n+1}$,
(40l)      $\psi_{n+1} := \omega_n\tilde{\delta}_{n+1}/(\tilde{\delta}_n\eta_m)$,
(40m)      $\tilde{s}_{n+1} := \tilde{r}_{n+1} - [\tilde{t}_n(1-\xi_m) + \tilde{s}_n\xi_m + A\tilde{s}_n\eta_m]\psi_{n+1}$
       endif;
(40n)  $\tilde{t}_{n+1} := \tilde{w}_{n+1} - \tilde{s}_n\psi_{n+1}$,
(40o)  $A\tilde{t}_{n+1} := A\tilde{w}_{n+1} - A\tilde{s}_n\psi_{n+1}$,
(40p)  $\varphi_{n+1} := y_0^H A\tilde{s}_{n+1}/\tilde{\delta}_{n+1}$.
```

If $\tilde{r}_{n+1} = o$, the process terminates and z_{n+1} is the solution of $Az = b$; if $\tilde{r}_{n+1} \neq o$ but $\tilde{\delta}_{n+1} = 0$ or $\varphi_{n+1} = 0$, the algorithm breaks down.

Like BiCGS and BiCGStab this algorithm requires two applications of A per step, i.e., one per degree of the residual polynomial, namely for $A\tilde{s}_n$ in (40c) and for $A\tilde{w}_{n+1}$ in (40d) or (40i). $A\tilde{t}_{n+1}$ is then obtained in (40o). In case of real data all the computation remains real. But even in the complex case BiCGStab2 is an improvement over BiCGStab since one performs a two-dimensional residual minimization in each other step. Of course, one could try to accomplish even higher dimensional minimizations in this framework, but clearly this would further complicate the algorithm and increase the memory requirement.

The methods BiCGStab and BiCGStab2 are examples of a class of methods that one might call *product methods* and which are characterized by residual polynomials that are the product of residual polynomials emerging from two different methods. Here, one factor is a Lanczos polynomial, while the other one comes from a one- or two-step minimal residual approach. Such product methods are truly *hybrid* methods, and deserve this name much more than many of the other approaches to which this notion has been applied. Note that the recursions of BiCGStab hold whenever the polynomials τ_n can be updated according to (16), and those of BiCGStab2 are valid whenever these polynomials satisfy (26). Only the definitions of the parameters χ_m, ξ_m, and η_m must be replaced.

For example, one might try a hybrid method that first applies another Krylov space method, say, GMRes, until it becomes too expensive; after computing a real factorization of the obtained residual polynomial τ_M of, say, degree M, one could from then on use this polynomial and its powers to create residuals of the form $r_{kM} = \rho_{(k-1)M}(A)\tau_M^k(A)r_0$ and intermediate ones where not yet all factors of τ_M appear with multiplicity k. The factored form of the GMRes residual τ_M yields recurrences (27) that can be reformulated to conform with (26). First experiments on this approach gave promising results, but not as good as those of BiCGStab2.

Moreover, if in generalization of (26) the polynomials τ_n satisfy a general three-term recursion that takes the condition $\tau_n(0) = 1$ into account, i.e., if

$$\tau_{n+1} := (1 - \xi_n)\tau_{n-1} + (\xi_n + \eta_n\zeta)\tau_n, \tag{41}$$

then the system of recurrences (28), (29), (32)–(34) is still valid if we replace χ_m, η_m by χ_n, η_n in (32)–(34). In the Krylov space these recurrences turn into (40c), (40b), (40i), (40m), and (40n), respectively, and they also yield (40j) and (40o). Consequently, (40a)–(40c) and (40i)–(40p) also yield a realization of a product method in which the Lanczos polynomials are combined with a sequence of polynomials τ_n satisfying a three-term recursion (41). For example, one could use shifted and scaled Chebyshev polynomials here to obtain a *Lanczos–Chebyshev method.*

It is known, see, e.g., [5], that the zeros of ρ_n and σ_n can be determined from the coefficients $\psi_1, \ldots, \psi_{n-1}$ and $\omega_0, \ldots, \omega_{n-1}$. In fact, if

$$L_n := \begin{bmatrix} \omega_0^{-1} & & & \\ -\omega_0^{-1} & \omega_1^{-1} & & \\ & \ddots & \ddots & \\ & & -\omega_{n-2}^{-1} & \omega_{n-1}^{-1} \end{bmatrix}, \quad R_n := \begin{bmatrix} 1 & \psi_1 & & \\ & 1 & \ddots & \\ & & \ddots & \psi_{n-1} \\ & & & 1 \end{bmatrix},$$

then the eigenvalues of the tridiagonal matrix L_nR_n are the zeros of ρ_n, and those of the $n \times n$ leading principal submatrix of the tridiagonal matrix $R_{n+1}L_{n+1}$ are the zeros of σ_n. The zeros of either of the two polynomials are often considered as approximations of eigenvalues of A. (This is, basically, the approach of the unsymmetric Lanczos algorithm for finding eigenvalues of A.) Experiments indicate, however, that the coefficients ψ_n and ω_n produced by BiCGS and BiCGStab are less accurate than those found by BiCG or by the Lanczos biorthogonalization algorithm (the "unsymmetric" Lanczos algorithm).

In BiCGStab2, the zeros $1/\zeta_j$ of the polynomials τ_n are, if $n = 2m$, the zeros of the quadratic factors

$$1 + (\eta_k - \chi_k\xi_k)\zeta - \eta_k\xi_k\zeta^2, \quad k = 0, \ldots, m-1. \tag{42}$$

These zeros can at best in a very vague way be considered as approximations to individual eigenvalues of A. Note in particular that they have no influence on the polynomials ρ_n and σ_n, and, a fortiori, on their zeros.

3. Numerical examples. First, we present several examples with non-Hermitian banded Toeplitz matrices of order $N = 200$. For these matrices, the behavior of the spectrum in the limit $N \to \infty$ is known [6], [13], but this is nearly irrelevant here. What counts for the convergence of iterative methods is the ε-pseudospectrum Λ_ε, which, when N is large and ε is small, is according to Reichel and Trefethen [9] approximately equal to the following union of three sets:

$$\Lambda_\varepsilon \equiv (\Lambda + \Delta_\varepsilon) \cup \Omega_r \cup \Omega^R. \tag{43}$$

Here, $\Lambda + \Delta_\varepsilon$ denotes the exact spectrum with disks of radius ε around each eigenvalue. To describe Ω_r and Ω^R, we need to look at the images $\phi(S_r)$ and $\phi(S_R)$ of the circles of radius r and R, respectively, under the mapping by the symbol ϕ of the Toeplitz matrix. Ω_r and Ω^R contain the points $\zeta \in \mathbb{C}$ with respect to which $\phi(S_r)$ and $\phi(S_R)$ have positive and negative, respectively, winding number. The radii $r < 1$ and $R > 1$ depend on ε and N according to $r := (\varepsilon/c)^{1/N}$ and $R := (\varepsilon/C)^{-1/N}$, where c and C are some constants, which for the plots in [9] have been set to 1. See Reichel and Trefethen [9] for details and for plots corresponding to some of our examples.

Banded Toeplitz matrices are of relevance in applications, since the discretization of partial differential equations often leads to such a matrix or a low-rank modification of one.

Example 1. Let us first consider the tridiagonal matrix

$$A := \begin{bmatrix} 4 & -2 & & \\ 1 & 4 & -2 & \\ & 1 & 4 & \ddots \\ & & \ddots & \ddots \end{bmatrix} \tag{44}$$

with the symbol $\phi(\zeta) = -2\zeta + 4 + \zeta^{-1}$. Its (unimportant) exact spectrum lies on the complex interval $[4 - 2i\sqrt{2}, 4 + 2i\sqrt{2}]$. The image $f(S_1)$, which is the boundary of the (continuous) spectrum of the associated Toeplitz operator with $N = \infty$ is an ellipse with foci $4 \pm 2i\sqrt{2}$, major semiaxis 3 and minor semiaxis 1. The ε-pseudospectrum is the interior of a slightly smaller ellipse with the same foci. Hence, Chebyshev iteration or second-order Richardson iteration, adapted to the family of ellipses with these foci, would have asymptotically optimal linear convergence. It is therefore no surprise that the generalized minimal residual method GMRES converges also approximately linearly.[2] As right-hand side b of the system $Az = b$ and for the initial vectors z_0 and y_0 we choose random vectors. (Of course, the same vectors are used for all the methods tested.)

In Fig. 1 we display the residual norm convergence of BIOMIN(= BICG), BIOMINSQ (= BICGS = CGS), BICGSTAB, the new algorithm BICGSTAB2, and a brute force implementation of GMRES(∞) (= GCR). The numbers on the x-axis give the iteration count, except for GMRES, where only every other iteration is counted. In GMRES each iteration requires just one application of A, in contrast to the other methods where multiplications both by A and A^H are needed; hence, it is fair to divide the numbers of iteration of GMRES by two, although, on the other hand, the long recurrences of GMRES(∞) become very expensive with respect to memory and arithmetic operations when n becomes large.

[2]The author is indebted to L. N. Trefethen for this interpretation of his numerical result.

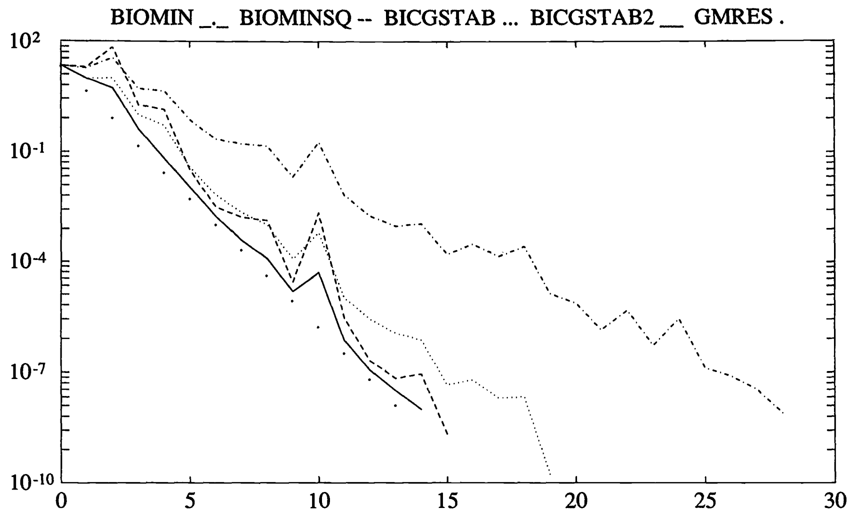


FIG. 1. *The residual norm history (i.e.,* $||r_n||/||r_0||$ *vs.* n*) for a linear system with the matrix* (44) *of order* 200, *solved by* BIOMIN = BICG *(dotted-dashed)*, BIOMINSQ = BICGS = CGS *(dashed)*, BICGSTAB *(dotted)*, BICGSTAB2 *(solid)*, *and* GMRES(∞) = GCR *(dotted). For* GMRES(∞) *only every other iteration is shown and counted, i.e.,* $||r_{2n}||/||r_0||$ *is plotted.*

In this example the convergence of all methods is rather fast, but it is interesting that the squared method BICGSTAB and BICGSTAB2 are considerably faster than BICG, and nearly as fast as GMRES(∞), which is optimal with respect to the measure of residual norm used in our figures. The same general behavior will be noted in our other examples. In this first example, where the matrix is real but the spectrum is complex, BICGSTAB2 is clearly better than the competing methods that require the same amount of work.

Example 2. As our second example we take

$$A := \begin{bmatrix} 2 & 1 & & & \\ 0 & 2 & 1 & & \\ 1 & 0 & 2 & 1 & \\ & 1 & 0 & 2 & \ddots \\ & & \ddots & \ddots & \ddots \end{bmatrix} \tag{45}$$

with the symbol $\phi(\zeta) = \zeta + 2 + \zeta^{-2}$. Its spectrum and pseudospectrum is three-fold rotationally symmetric with respect to its center at $\zeta = 2$; see Fig. 8 in [9] for a plot. This symmetry increases the chance of breakdown in the Lanczos process, and we noticed in fact many breakdowns in examples of this type. (Breakdowns also depend on the initial vectors z_0 and y_0, but since these are chosen real for this real matrix, they are not really in general position with respect to the eigenvectors.) In this particular example, BICGSTAB broke down in step 26.

Our results are shown in Fig. 2. The convergence is slower and rougher than in Example 1, but the relative performance of the various methods remains the same.

Example 3. While both previous examples are real matrices, we consider now the complex Toeplitz matrix

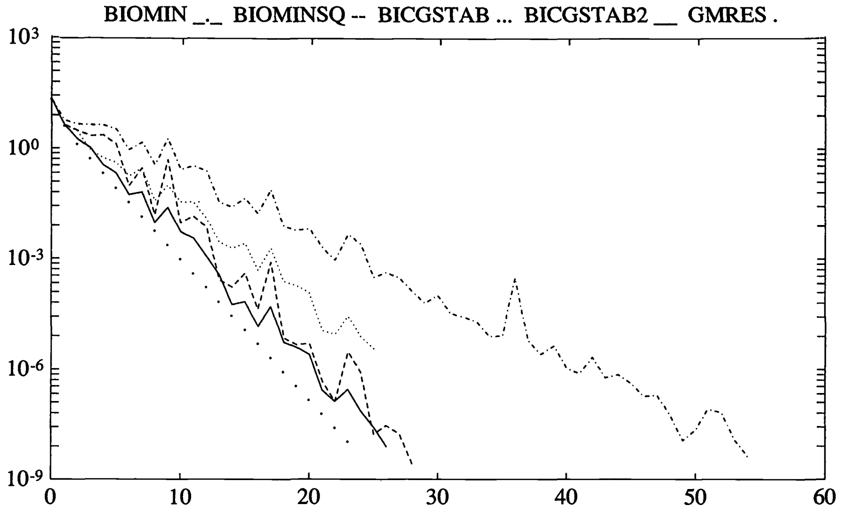


FIG. 2. *The residual norm history for a linear system with the real Toeplitz matrix* (45) *of order* 200. *The selection of methods and the labels of the axes are the same as in Fig.* 1.

$$A := \begin{bmatrix} 4 & 0 & 1 & .7 & & \\ 2i & 4 & 0 & 1 & .7 & \\ & 2i & 4 & 0 & 1 & \ddots \\ & & 2i & 4 & 0 & \ddots \\ & & & 2i & 4 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix} \tag{46}$$

with the symbol $\phi(\zeta) = .7\zeta^3 + \zeta^2 + 4 + 2i\zeta^{-1}$. Its pseudospectrum plot is interpreted as "Picasso's head of a bull" by Reichel and Trefethen; see their Fig. 7 in [9]. Our results are shown in Fig. 3. Both our complex BiCGStab and the (also complex) BiCGStab2 do very well, but the difference in their behavior is now, not unexpectedly, much smaller. Again, one needs to point out that the dots for GMRes(∞) represent the optimal residual norm convergence, but that the computational and, in particular, the memory requirements for this method are considerably higher.

Example 4. We finally consider a "real world" example from the Harwell–Boeing collection of large sparse test matrices, namely OILGEN1, a matrix of order 2205 with 14'133 nonzeros, which comes from an oil reservoir simulation on a $21 \times 21 \times 5$ grid. Figure 4 shows the convergence history for BiCGStab and BiCGStab2, both applied *without* preconditioning. BiCG and BiCGS were also tried, but the former converges again only about half as fast (requiring more than 200 iterations to reduce the relative residual to 10^{-5}) and the latter has so many steep peaks in its residual convergence history that the curve would cover up most of what is now shown in Fig. 4. In this difficult example these methods are no longer able to produce a smooth convergence curve, but they do much better than BiCG and BiCGS.

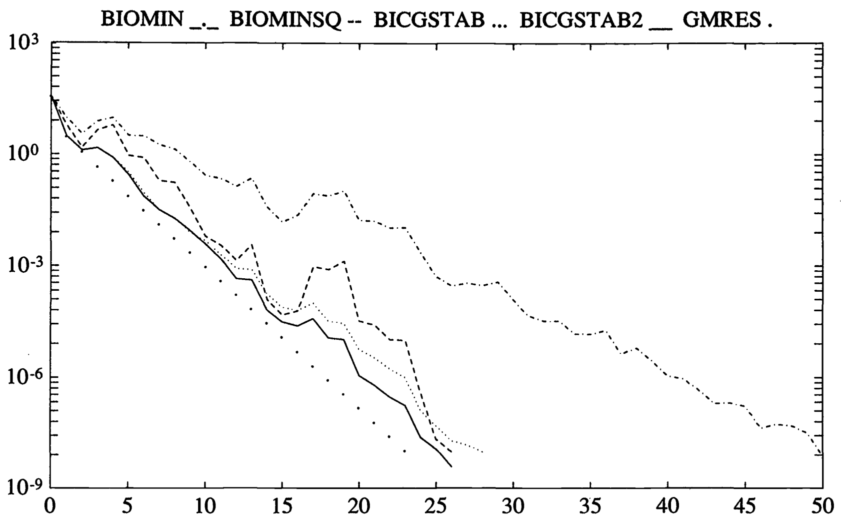


FIG. 3. *The residual norm history for a linear system with the complex Toeplitz matrix* (46) *of order* 200. *The selection of methods and the labels of the axes are the same as in Fig.* 1.

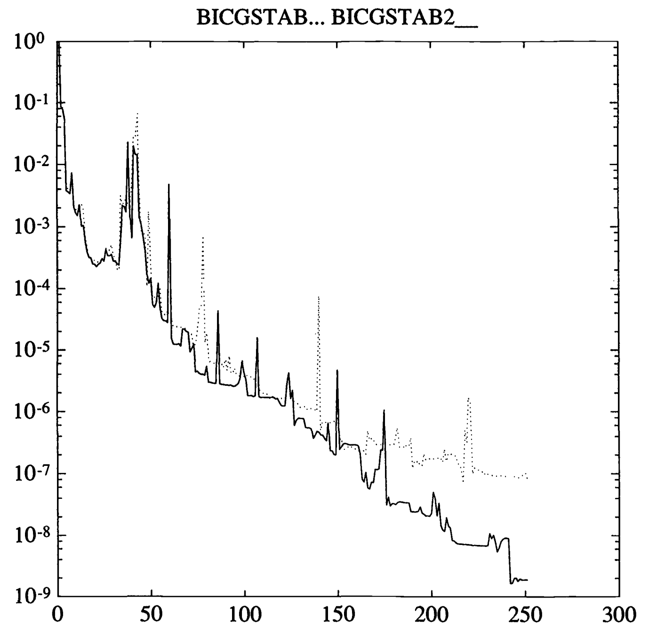


FIG. 4. *The residual norm history* (*i.e.*, $||r_n||/||r_0||$ *vs.* n) *for a linear system with the* OILGEN1 *matrix of the Harwell-Boeing collection, solved with* BiCGSTAB (*dotted*) *and* BiCGSTAB2 (*solid*).

Acknowledgment. The author would like to thank Noël Nachtigal for running Example 4 on a test installation of MATLAB 4.

Note added in proof. The algorithm BiCGSTAB2 introduced here should not be considered as a black box solver for sparse linear systems. As we have pointed out, there are situations where BiCGStab2 (as well as BiCG and BiCGStab) can break down or become unstable. A reliable program would have to be able to take appropriate measures in these situations. There are also cases where single steps of BiCGStab should be executed between some of the (double) steps of BiCGStab2; in the present version of the latter, the zeros of the intermediate steps are always dismissed, and this can be a disadvantage.

REFERENCES

[1] R. Fletcher, *Conjugate gradient methods for indefinite systems*, in Numerical Analysis, G. A. Watson, ed., Vol. 506, Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1976, pp. 73–89.

[2] R. Freund, M. Gutknecht, and N. Nachtigal, *An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices*, SIAM J. Sci. Comput., 14 (1993), pp. 137–158.

[3] M. H. Gutknecht, *A completed theory of the unsymmetric Lanczos process and related algorithms*, Part I, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 594–639.

[4] ———, *A completed theory of the unsymmetric Lanczos process and related algorithms*, Part II, SIAM J. Matrix Anal. Appl., to appear.

[5] ———, *The unsymmetric Lanczos algorithms and their relations to Padé approximation, continued fractions, and the* qd *algorithm*, Preliminary Proc. Copper Mountain Conference on Iterative Methods (preliminary version), April 2-5, 1990.

[6] I. I. Hirschman, jr., *The spectra of certain Toeplitz matrices*, Illinois J. Math., 11 (1967), pp. 145–159.

[7] C. Lanczos, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bureau Standards, 49 (1952), pp. 33–53.

[8] 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.

[9] L. Reichel and L. N. Trefethen, *Eigenvalues and pseudo-eigenvalues of Toeplitz matrices*, Linear Algebra Appl., 162–164 (1992), pp. 153–185.

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

[11] P. Sonneveld, CGS, *a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[12] L. N. Trefethen, *Non-normal matrices and pseudo-eigenvalues*, manuscript.

[13] J. L. Ullman, *Toeplitz matrices associated with a semi-infinite Laurent series*, Bull. Amer. Math. Soc., 73 (1967), pp. 883–885.

[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.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1034–1056, September 1993 003

BLOCK SPARSE CHOLESKY ALGORITHMS ON ADVANCED UNIPROCESSOR COMPUTERS*

ESMOND G. NG† AND BARRY W. PEYTON†

Abstract. As with many other linear algebra algorithms, devising a portable implementation of sparse Cholesky factorization that performs well on the broad range of computer architectures currently available is a formidable challenge. Even after limiting the attention to machines with only one processor, as has been done in this paper, there are still several interesting issues to consider. For dense matrices, it is well known that block factorization algorithms are the best means of achieving this goal. This approach is taken for sparse factorization as well.

This paper has two primary goals. First, two sparse Cholesky factorization algorithms, the multifrontal method and a blocked left-looking sparse Cholesky method, are examined in a systematic and consistent fashion, both to illustrate the strengths of the blocking techniques in general and to obtain a fair evaluation of the two approaches. Second, the impact of various implementation techniques on time and storage efficiency is assessed, paying particularly close attention to the work-storage requirement of the two methods and their variants.

Key words. sparse linear systems, Cholesky factorization, supernodes, block algorithms, advanced computer architectures

AMS subject classifications. 65F, 65W

1. Introduction. Many scientific and engineering applications require the solution of large sparse symmetric positive definite systems of linear equations. *Direct* methods use Cholesky factorization followed by forward and backward triangular solutions to solve such systems. For any $n \times n$ symmetric positive definite matrix A, its Cholesky factor L is the lower triangular matrix with positive diagonal such that $A = LL^T$. When A is sparse, it will generally suffer some fill during the computation of L; that is, some of the zero elements in A will become nonzero elements in L. In order to reduce time and storage requirements, only the nonzero positions of L are stored and operated on during *sparse* Cholesky factorization. Techniques for accomplishing this task and for reducing fill have been studied extensively (see [12], [19] for details). In this paper we restrict our attention to the numerical factorization phase. We assume that the preprocessing steps, such as reordering to reduce fill and symbolic factorization to set up the compact data structure for L, have been performed. Details on the preprocessing can be found in [12], [19].

As with many other linear algebra algorithms, devising a portable implementation of sparse Cholesky factorization that performs well on the broad range of computer architectures currently available is a formidable challenge. Even after limiting our attention to machines with only one processor, as we have done herein, there are still several interesting issues to consider. In this paper we will investigate sparse Cholesky algorithms designed to run efficiently on vector supercomputers (e.g., the Cray Y-MP) and on powerful scientific workstations (e.g., the IBM RS/6000, the DEC 5000, and the Stardent P3000). To achieve high performance on such machines, the algorithms must be able to exploit vector processors and/or pipelined functional units. Moreover, with the dramatic increases in processor speed during the past few years, rapid memory access has become a very important factor in determining performance levels on several of these

* Received by the editors November 26, 1991; accepted for publication (in revised form) June 10, 1992. This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

† Mathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, Tennessee 37831-6367.

machines. To be efficient, algorithms must reuse data in fast memory (e.g., cache) as much as possible. Consequently, a highly localized and regular memory-access pattern is ideal for many of today's fastest machines.

It is well known that block factorization algorithms are the best means of achieving this goal. Perhaps the best-known example of a software package based on this approach is LAPACK, a software package for performing dense linear algebra computations on advanced computer architectures including shared-memory multiprocessor systems [2]. Each block algorithm in LAPACK is built around some computationally intensive variant of a matrix-matrix (BLAS3) or matrix-vector (BLAS2) multiplication kernel subroutine, which can be optimized for each computing platform on which the package is run.

The sparse block Cholesky algorithms discussed in this paper take essentially the same approach; we do not, however, include multiprocessors nor do we tune the kernels for efficiency on specific machines. We investigate two algorithms:

1. The multifrontal method [15], [24], which is based on the right-looking formulation of the Cholesky factorization algorithm.
2. A left-looking block algorithm that has, until recently, received little attention in the literature [28].

Both methods will use the same kernel subroutines to do all the numerical work required during the factorization. The differences are limited to such issues as:

- indirect addressing and other integer operations related to the *structural* aspects of sparse factorization,
- the ability to reuse data in cache,
- the amount of data movement,
- the memory-access pattern, and
- the working storage requirement.

In general, variations in the efficiency of the block algorithms and their variants are not very large. However, our tests indicate significant differences in the amount of working storage and expensive data movement required.

This paper has two primary goals. First, we will look at the two block Cholesky factorization algorithms in a systematic and consistent fashion, both to illustrate the strengths of the blocking techniques in sparse matrix computations in general and to obtain a fair evaluation of the two basic approaches. Second, we will assess the value of various implementation techniques on time and storage efficiency, paying particularly close attention to the working storage requirement of the two methods and their variants.

Rothberg and Gupta [28] have studied these algorithms independently. They consider the caching issue in more detail and implement a more complicated and effective loop-unrolling scheme than we do. However, they do not compare the working storage requirements of the various algorithms as we do. We have introduced enhancements to the multifrontal algorithm that greatly reduce the amount of stack storage and data movement overhead required by that algorithm. Also, we consider the performance of these algorithms on a vector supercomputer and a high-performance workstation with vector hardware.

This paper is organized as follows. Section 2 contains notation and other background material needed to present the algorithms, including a discussion of previous work on block sparse Cholesky algorithms. Section 3 describes the left-looking block Cholesky algorithm and some of its key features. Presented in §4 are implementation details and enhancements for both the left-looking block algorithm and the multifrontal algorithm. Section 5 contains the results of our performance tests on several of the

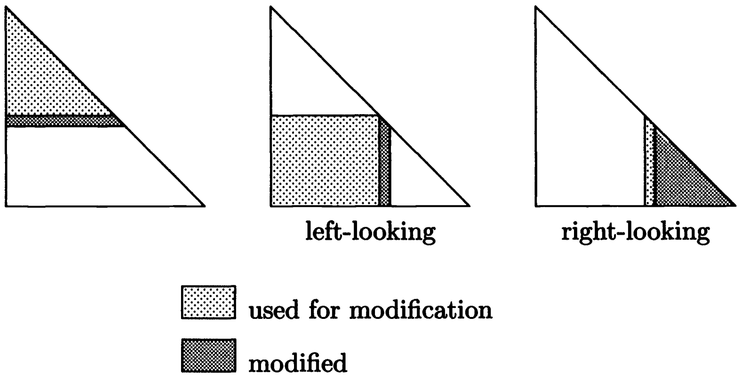


FIG. 1. *Three forms of Cholesky factorization.*

machines mentioned earlier in this section. Finally, concluding remarks and speculations on future work appear in §6.

2. Background material.

2.1. Column-based Cholesky factorization methods.

The bulk of the work in Cholesky factorization of a symmetric positive definite matrix A occurs in a triply nested loop around the single statement

$$A_{ij} = A_{ij} - (A_{ik}A_{kj})/A_{kk}.$$

By varying the order in which the loop indices i, j, and k are nested, we obtain three different formulations of Cholesky factorization, each with a different memory access pattern.

1. *Bordering Cholesky*. Taking i in the outer loop, successive rows of L are computed one by one, with the inner loops solving a triangular system for each new row in terms of the previously computed rows (see Fig. 1).
2. *Left-looking Cholesky*. Taking j in the outer loop, successive columns of L are computed one by one, with the inner loops computing a matrix-vector product that gives the effect of previously computed columns on the column currently being computed.
3. *Right-looking Cholesky*. Taking k in the outer loop, successive columns of L are computed one by one, with the inner loops applying the current column as a rank-1 update to the remaining partially reduced submatrix.

The various versions of Cholesky factorization can be used to take better advantage of particular architectural features of a given machine (cache, virtual memory, vectorization, etc.) [11]. For more details concerning these three versions of Cholesky factorization, consult George and Liu [19, pp. 18–21].

The bordering method requires a row-oriented data structure for storing the nonzeros of L. Liu [25] has devised a compact row-oriented data structure for this purpose, but currently the technique has not been successfully adapted to run efficiently on modern workstations and vector supercomputers. Consequently, our paper will focus on block versions of the left-looking and right-looking algorithms (also known as column-Cholesky and submatrix-Cholesky, respectively). Both the left-looking and right-looking algorithms naturally require a column-oriented data structure, which is easy to construct [31]. Thus, we restrict our attention to column-oriented implementations of the left-looking and right-looking algorithms.

We need the following definitions to write down the algorithms. Let M be an $n \times n$ matrix, and denote the jth column of M by $M_{*,j}$. The sparsity structure of column j in the *lower triangular* part of M is denoted by $Struct(M_{*,j})$. That is,

$$Struct(M_{*,j}) := \{s \geq j : M_{s,j} \neq 0\}.$$

Column-oriented Cholesky factorization algorithms can be expressed in terms of the following two subtasks:

1. $cmod(j, k)$: modification of column j by a multiple of column k, $k < j$,
2. $cdiv(j)$: division of column j by a scalar.

Of course, sparsity in columns j and k is exploited when A and L are sparse. Using these basic operations, Figs. 2 and 3 give high-level descriptions of the basic left-looking and right-looking sparse Cholesky factorization algorithms, respectively. (We will refer to these two algorithms as left-looking and right-looking `col-col`.)

```
for $j = 1$ to $n$ do
   for $k$ such that $L_{j,k} \neq 0$ do
      $cmod(j, k)$
   $cdiv(j)$
```

FIG. 2. *Left-looking sparse Cholesky factorization algorithm* (*left-looking* `col-col`).

```
for $k = 1$ to $n$ do
   $cdiv(k)$
   for $j$ such that $L_{j,k} \neq 0$ do
      $cmod(j, k)$
```

FIG. 3. *Right-looking sparse Cholesky factorization algorithm* (*right-looking* `col-col`).

Left-looking sparse Cholesky is the simpler of the two algorithms to implement, and it appears in several well-known commercially available sparse matrix packages [8], [16]. For implementation details, the reader should consult George and Liu [19]. Straightforward implementations of the right-looking approach are generally quite inefficient because matching the updating column k's sparsity pattern with that of each column j in the updated submatrix requires expensive searching through the row indices in $Struct(L_{*,k})$ and $Struct(L_{*,j})$, $j \in Struct(L_{*,k}) - \{k\}$. Consequently, we will not pursue such an implementation in this paper. However, Rothberg and Gupta [28] have recently reported that a block version of this approach is reasonably competitive, because for practical problems the blocking greatly reduces the amount of index matching needed. Note also that a straightforward implementation of the right-looking approach forms the basis for a distributed-memory parallel factorization algorithm known as the *fan-out method* [6], [18], [20]. In this paper we will study a left-looking block algorithm and also the *multifrontal* algorithm [15], [24], which can be viewed as an efficient implementation of right-looking sparse Cholesky factorization as we shall see in §2.3.

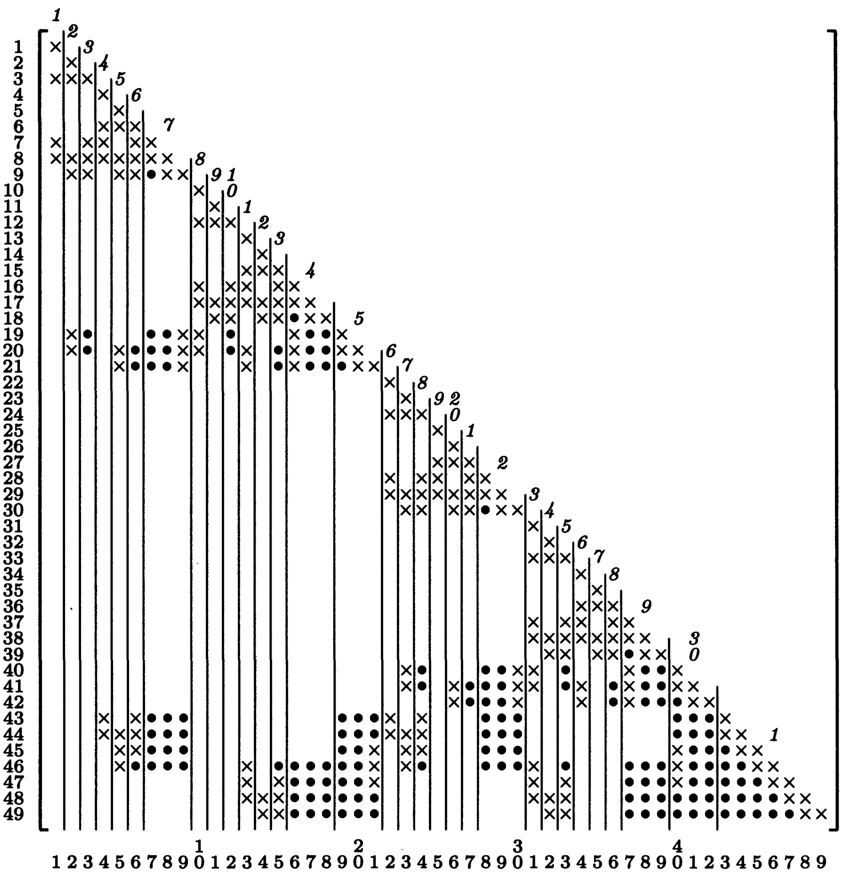


FIG. 4. *Supernodes for* 7 × 7 *nine-point grid problem ordered by nested dissection.* (× *and* • *refer to nonzeros in A and fill in L, respectively. Numbers over diagonal entries label supernodes.*)

2.2. Supernodes and elimination trees. Efficient implementations of both the multifrontal algorithm and left-looking block algorithms require that columns of the Cholesky factor L sharing the same sparsity structure be grouped together into *supernodes*. More formally, the set of contiguous columns[1] $j, j+1, \ldots, j+t$ constitutes a supernode if $Struct(L_{*,k}) = Struct(L_{*,k+1}) \cup \{k\}$ for $j \le k \le j+t-1$. Most commonly used in practice is the set of *fundamental supernodes* associated with the factor L. Each fundamental supernode satisfies the following additional requirement: It is a maximal supernode for which the first column is the only column of the supernode that can have more than one child in the elimination tree associated with L. (Elimination trees are defined in the next paragraph.) The fundamental supernodes for an example matrix is shown in Fig. 4.

Note that the columns of a supernode $\{j, j+1, \ldots, j+t\}$ have a dense diagonal block and have *identical* column structure below row $j+t$. Note also that columns in the same supernode can be treated as a unit for both computation and storage. (See, for example, [26] for further details.)

[1]It is convenient to denote a column $L_{*,j}$ belonging to a supernode by its column index j. It should be clear by context when j is being used in this manner.

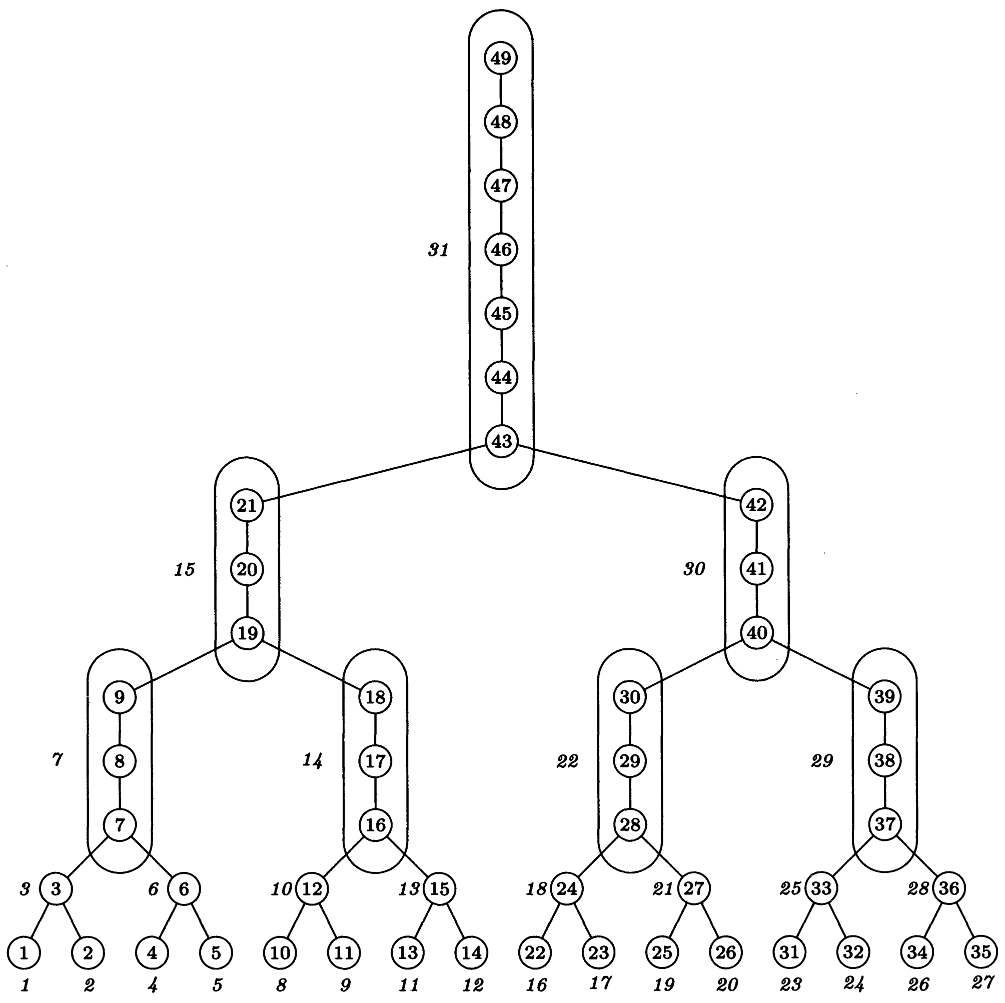


FIG. 5. *Elimination tree (and supernode elimination tree) for the matrix shown in Fig. 4. Ovals enclose supernodes that contain more than one node. Nodes not enclosed by an oval are singleton supernodes. Italicized numbers label supernodes.*

The multifrontal method makes explicit use of the elimination tree associated with L. For each column $L_{*,j}$ having off-diagonal nonzero elements, we define the *parent* of j to be the row index of the first off-diagonal nonzero in that column. For example, the parent of node 9 is node 19 for the matrix in Fig. 4. It is easy to see that the parents of the columns define a tree structure, which is called the *elimination tree* of L. Associated with any supernode partition is a *supernodal elimination tree*, which is obtained from the elimination tree essentially by collapsing the nodes (columns) in each supernode into a single node (block column). This can be done because the nodes in each supernode form a chain in the elimination tree. Fig. 5 (see above) displays the elimination tree for the matrix in Fig. 4. The supernodal elimination tree for the partition in Fig. 4 is also shown in Fig. 5, superimposed on the underlying elimination tree. Throughout this paper we use N to denote the number of supernodes in L.

2.3. Supernode-based Cholesky factorization algorithms: previous work. Figures 2 and 3 contain high-level descriptions of sparse Cholesky algorithms whose innermost loop updates a single column j with a multiple of a single column k. The next two subsections briefly describe two well-known sparse Cholesky algorithms that exploit the shared sparsity structure within supernodes to improve performance. The first is the left-looking `sup-col` algorithm, whose atomic operation is updating the target column j with every column in a supernode not containing j or with a subset of columns in the supernode containing j (in either case, a BLAS2 operation). The other is the more widely known multifrontal method.

2.3.1. Left-looking `sup-col` Cholesky factorization. The basic idea behind the left-looking `sup-col` Cholesky algorithm is very simple. Let $K = \{p, p+1, \ldots, p+q\}$ be a supernode[2] in L and consider the computation of $L_{*,j}$ for some $j > p+q$. It follows from the definition of supernodes that column $A_{*,j}$ will be modified by *no* columns of K or *every* column of K. Previous studies [7], [26], [27], [29] have demonstrated that this observation has important ramifications for the performance of left-looking sparse Cholesky factorization. Loosely speaking, when used to update the target column $L_{*,j}$, the columns in a supernode K can now be treated as a single unit (or *block column*) in the computation. Since the columns in a supernode share the same sparsity structure below the dense diagonal block, modification of a particular column $j > p+q$ by these columns can be accumulated in a work vector using *dense* vector operations, and then applied to the target column using a single *sparse* vector operation that employs indirect addressing. Moreover, the use of *loop unrolling* in the accumulation, as described in [10], reduces memory traffic.

In Fig. 6 we present the left-looking `sup-col` Cholesky factorization algorithm. The reader will find a more detailed implementation of the algorithm presented in [26]. In order to keep the notation simple, K is to be interpreted in one of two different senses,

```
for $J = \mathit{1}$ to $N$ do
    Scatter $J$'s relative indices into indmap.
    for $j \in J$ (in order) do
        for $K$ such that $L_{j,K} \neq 0$ do
            $t \leftarrow cmod(j, K)$
            Assemble $t$ into $L_{*,j}$ using indmap.
        $cmod(j, J)$
        $cdiv(j)$
```

FIG. 6. *Left-looking* `sup-col` *Cholesky factorization algorithm.*

depending on the context in which it appears. In one context (e.g., line 3 of Fig. 6), K is interpreted as the set of columns in the supernode, i.e., $K = \{p, p+1, \ldots, p+q\}$. In the first line of the algorithm, the supernodes are treated as an ordered set of loop indices $\mathit{1}, \mathit{2}, \ldots, K, \ldots, N$, where $K < J$ if and only if $p < p'$, where p and p' are the first columns of K and J, respectively. This dual-purpose notation is also illustrated in Fig. 4, where the supernode labels are written over the diagonal entries, yet we can still write $\mathit{30} = \{40, 41, 42\}$, for example. We denote both the last supernode and the number of supernodes by N.

[2]Throughout the remainder of the paper the numbers designating a supernode will be italicized, and the letters denoting a supernode will be capitalized.

Suppose $K = \{p, p+1, \ldots, p+q\}$. Whenever $j > p+q$ and $L_{j,p+q} \neq 0$, the task *cmod*(j, K) consists of the operations *cmod*(j, k), where $k = p, p+1, \ldots, p+q$. When $j \in K$, *cmod*(j, K) consists of the operations *cmod*(j, k), for $k = p, p+1, \ldots, j-1$. We let $L_{j,K}$ denote the 1 by $|K|$ submatrix in L induced by row j and the columns in K.

The indirect addressing scheme used by the algorithm is as follows. The indices in each list $Struct(L_{*,j})$ are sorted in ascending order during the preprocessing stage (i.e., symbolic factorization). For each row index $i \in Struct(L_{*,j})$, the corresponding *relative index* is the position ℓ of i relative to the bottom of the list. For example, $\ell = 0$ for the last index in the list, $\ell = 1$ for the next-to-the-last one, and so forth. For each supernode $J = \{p, p+1, \ldots, p+q\}$, define

$$Struct(L_{*,J}) = Struct(L_{*,p}).$$

The relative position ℓ of each row index $i \in Struct(L_{*,J})$ is stored in an n-vector *indmap* as follows: *indmap*$[i] \leftarrow \ell$. First the update *cmod*(j, K) is accumulated in a work vector t whose length is the number of nonzero entries in the update. That is, the update is computed and stored as a *dense* vector t. Then the algorithm assembles (scatter-adds) t into factor storage, using *indmap*$[i]$ to map each active row index $i \in Struct(L_{*,K})$ to the appropriate location in $L_{*,j}$ to which the corresponding component of t is added. The notion of relative indices apparently was first proposed by Schreiber [30].

2.3.2. Multifrontal Cholesky factorization. The multifrontal method, introduced by Duff and Reid in [15], is well documented in the literature. With much of its work performed within dense frontal matrices, this method has proven to be extremely effective on vector supercomputers [1], [3], [7], [9]. Moreover, the multifrontal method is naturally expressed and implemented as a block method, and several of the advantages it derives from block matrix operations have already been explored in the literature: e.g., its ability to reuse data in fast memory [1], [29] and its ability to perform well on machines with virtual memory and paging [23]. Implementation of the multifrontal method is more complicated and involves more subtleties than do any of the left-looking Cholesky variants. For the purposes of this paper, it is adequate to restrict our presentation to an informal outline of the method. For a detailed survey of the multifrontal method and the techniques required for an efficient implementation, the reader should consult Liu [24]. The following paragraphs discuss the informal statement of the algorithm, found in Fig. 7.

The outer loop of the supernodal multifrontal algorithm processes the supernodes $1, 2, \ldots, N$, where the supernodes have been renumbered by a postorder traversal of the supernodal elimination tree. After moving the required columns of $A_{*,J}$ into the leading columns of J's dense *frontal matrix* F_J, the algorithm pops from the *update matrix stack* an update matrix U_K for each child K of J in the supernodal elimination tree and assembles these accumulated update columns into F_J. (The postordering enables the use of a simple and efficient stack for the update matrices.) The update matrix U_K is a dense matrix containing *all* updates destined for ancestors of K from columns in the subtree of the supernodal elimination tree rooted at K. The assembly operation adds each entry of U_K to the corresponding entry of F_J. These are sparse operations requiring indirect indexing because an update matrix generally modifies a *proper subset* of the entries in the target frontal matrix. These are the only sparse operations required by the multifrontal method.

Now with all the necessary data accumulated in F_J, the next step in the main loop applies dense left-looking Cholesky factorization to the first $|J|$ columns in F_J (which we will call a *cdiv*(J) operation) to compute the block column $L_{*,J}$ and then accumulates

```
Initialize the update matrix stack.
for $J = 1$ to $N$ (in postorder) do
    Move $A_{*,J}$ into $F_J$.
    for each child $K$ of $J$ on top of the stack do
        Pop $U_K$ from stack.
        Assemble $U_K$ into $F_J$.
    Within $F_J$,
        compute the columns of $L_{*,J}$ ($cdiv(J)$),
        and compute all update columns from $L_{*,J}$
        (i.e., $cmod(k, J)$, where $k \in Struct(L_{*,J}) - J$).
    Move the new factor columns from $F_J$ to $L_{*,J}$.
    Move $U_J$ to the top of the stack.
```

FIG. 7. *Supernodal multifrontal Cholesky factorization algorithm.*

in the trailing columns of F_J all column updates $cmod(k, j)$, where $j \in J$ and $k \in Struct(L_{*,J}) - J$. At this point, the leading $|J|$ columns of F_J contain the columns of $L_{*,J}$, and the other columns have accumulated every update column for ancestors of J contributed by J and its descendants in the supernodal elimination tree. The algorithm then moves the newly computed columns to the appropriate location in the data structure for L, moves the update matrix U_J down onto the top of the stack, and proceeds with the next step of the major loop.

Three issues will occupy our attention when we take up the multifrontal algorithm again in §4. First, since all updates from the columns in $L_{*,J}$ are computed immediately after the new factor columns are computed, the multifrontal method provides the opportunity for optimal reuse of columns loaded in cache. Second, the costs of data movement overhead are potentially significant. We are referring here to the movement of matrix columns between each frontal matrix and L's data structure, and the movement of each update matrix from the location in working storage, where it was computed, to its storage-saving location at the top of the stack. This issue is of particular concern on machines with cache, where moving large amounts of data in this manner will cause expensive cache misses not incurred by the left-looking algorithms. Third, we will be concerned with the amount of storage required for the stack of update matrices, an issue that has received considerable attention in past studies [3], [22], [24].

3. Left-looking sup–sup Cholesky factorization. The idea behind the left-looking sup–sup Cholesky factorization algorithm is simple: The $cmod(j, K)$ operation is blocked one level higher, creating a supernode-to-supernode block-column updating operation $cmod(J, K)$ around which the new algorithm is constructed. The $cmod(J, K)$ operation performs $cmod(j, K)$ for every column $j \in J$ updated by the columns of K (a BLAS3 operation). The idea of constructing a sparse Cholesky algorithm around this operation is not new. Ashcraft and Peyton wrote a left-looking sup–sup sparse Cholesky factorization code, which was mentioned in [7], but was not presented there. The indexing scheme they used, however, was unnecessarily complex. Though efficient, it had the side effect of destroying the row indices of the nonzeros in L so that they had to be recomputed later for use during the triangular solution phase or any future factorizations of matrices with the same structure. For these reasons, they ultimately concluded that their implementation was unacceptable. Ashcraft recently sketched out a high-level version of the algorithm in a report on a different topic [4]. He has also created a single-parameter hybrid sparse Cholesky algorithm that performs a left-looking sup-sup factorization when the parameter takes on one extreme value and performs a supernodal multifrontal factorization when it takes on the opposite extreme value [5]. The left-looking sup-sup approach was proposed again by Ng and Peyton [26] as a promising candidate for parallelization on shared-memory multiprocessors. Parts of this work are steps toward completing the goals stated in the conclusion of that report. Recently and independently, Rothberg and Gupta examined the caching behavior of three block Cholesky factorization algorithms, including the multifrontal and left-looking sup-sup methods [28].

The following paragraphs discuss the left-looking sup-sup Cholesky factorization algorithm in Fig. 8 and its more basic implementation issues. One new item of notation is introduced; we let $L_{J,K}$ denote the $|J|$ by $|K|$ submatrix in L induced by the members of J and the members of K.

```
for $J = 1$ to $N$ do
    Scatter $J$'s relative indices into indmap.
    for $K$ such that $L_{J,K} \neq 0$ do
        Compute the number of columns of $J$ to be updated by the columns of $K$.
        $T \leftarrow cmod(J, K)$
        Gather $K$'s indices relative to $J$'s structure from indmap into relind.
        Using relind, assemble $T$ into $L_{*,J}$.
    $cdiv(J)$
```

FIG. 8. *Left-looking* sup-sup *Cholesky factorization algorithm.*

The bulk of the work is performed within the *cmod*(J, K) and *cdiv*(J) operations. The underlying matrix–matrix multiplication subroutine, which performs most of the work in the implementation, is used by the block multifrontal code as well, enabling a fair comparison of the two approaches. As in the left-looking sup-col approach, the update columns are accumulated in working storage. Naturally, far more working storage is required to accumulate the *cmod*(J, K) updates than is required to accumulate the *cmod*(j, K) updates, which consists of a single dense column no larger than the column of L with the most nonzero entries. This storage overhead will receive further attention in §§4 and 5.

Another distinction between the left-looking sup-col and sup-sup algorithms is that the sup-sup algorithm must compute the number of columns of J to be updated by the columns of K, which it does by searching for all row indices $i \in J \cap Struct(L_{*,K})$ in K's sorted index list.

The algorithm handles indirect addressing in much the same way that the sup-col algorithm in Fig. 6 does, with one key difference that generally improves its efficiency. (See §2.3.1 for other details about the indexing scheme.) The sup-sup algorithm gathers the indices of *K relative to J* from *indmap* into a temporary vector *relind*: Each active row index $i \in Struct(L_{*,K})$ is replaced by *indmap*$[i]$ in the integer vector *relind*. This single gather operation provides the indexing information for assembling the entire block update into factor storage (i.e., the storage that will contain $L_{*,J}$). The sup-col algorithm essentially has to repeat this gather operation each time it assembles a *cmod*(j, K) update ($j \in J$) into factor storage.

4. Implementation details and options. Section 5 reports performance statistics for implementations of the multifrontal and the left-looking `sup-sup` Cholesky factorization algorithms on several powerful uniprocessor computing systems. Our Fortran codes have not been tuned for performance on any *specific machine* except for our choice of the level of loop unrolling. To run efficiently on some of these machines, however, our implementations cannot afford to ignore other architectural considerations altogether. Unless they make effective use of data (i.e., columns of the matrix) once they have been loaded into cache, their performance will be severely penalized by an excessive number of cache misses. Thus, our implementations must be designed with this goal in mind. Our codes need to know the cache size on each machine to reuse cached data effectively (see §4.1). The cache size and the level of loop unrolling are the *only* machine-dependent parameters in our codes. Other implementation options and enhancements, which are entirely independent of the computer architecture, are also discussed in this section.

4.1. Reuse of data in cache. Consider the computation of a *cmod*(J, K) update during the left-looking `sup-sup` Cholesky factorization. Suppose the operation updates q columns of J with the columns of K. The number of columns updated may be as few as 1 or as many as $|J|$. We can compute *cmod*(J, K) as a sequence of `sup-col` updates *cmod*(j, K) for the q columns $j \in J$. If the columns of K, which happen to be stored contiguously in main memory, fit into cache memory, then the first *cmod*(j, K) loads the columns of K into cache, while the following $q - 1$ *cmods* will have extremely fast access to this data because it is already in cache.

Quite often, however, the columns of a supernode do not fit into the 32K or 64K caches used on current workstations. This can dramatically increase the number of cache misses associated with the final $q - 1$ *cmods*, as the columns of K overwrite one another as they are repeatedly read into cache. To avoid this problem, the algorithm partitions large supernodes into "panels" of contiguous columns that fit into the cache, as Rothberg and Gupta have done in their studies [27]–[29]. If K has been partitioned into two panels, then the *cmod*(J, K) update is performed by applying the *cmods* from the first panel to the q target columns of J, then applying the *cmods* from the second panel to the q target columns of J. We use essentially the same strategy to increase the reuse of data in cache by our multifrontal codes. This simple strategy has proven effective for the problems, machines, and factorization methods used in our tests. Extremely large problems, however, may require more complicated techniques that involve both horizontal and vertical partitioning and perhaps even sweeping changes in the data structure used to store L. The reader should consult Rothberg and Gupta [28] for a thorough discussion of these and many other issues associated with improving reuse of data in cache by both the multifrontal and the left-looking `sup-sup` sparse Cholesky algorithms.

4.2. Traversing row-structure sets. The left-looking `col-col` algorithm needs access to the row-structure sets $\mathcal{R}_j = \{k : L_{j,k} \neq 0\}$ (see Fig. 2). These row-structure sets must be computed from or traversed within the strictly column-oriented data structure used by the algorithm. By far the most commonly used method is to maintain the row-structure sets as linked lists within a single integer n-vector. Every column belongs to one and only one row-structure list at any given time during the course of the factorization. After a column update is completed, the column is placed in the list belonging to the next column it will modify. Details of this approach can be found in George and Liu [19, pp. 152–155] and in Ng and Peyton [26]. The same technique applies to the row-structure sets for the left-looking `sup-col` and `sup-sup` algorithms.

There is another way to determine the row-structure sets in the left-looking `col-col` algorithm, which relies on the fact that each row-structure set $\mathcal{R}_j$ is a pruned subtree of the elimination tree [21], [30]. Consequently, if the elimination tree is made available to the factorization algorithm, each member of $\mathcal{R}_j$ can be visited by performing a depth-first traversal of the appropriate pruned subtree. Implementation details can be found in Schreiber [30]. Again, the same technique applies to the row-structure sets for the left-looking `sup-col` and `sup-sup` algorithms. This approach is particularly attractive in a parallel implementation of the left-looking factorization algorithms for shared-memory multiprocessor systems since it eliminates the need for critical sections when manipulating the row-structure sets. We are currently pursuing this idea.

For the `sup-col` algorithm, our tests indicate that the total factorization time using the tree-traversal technique is slightly larger than that using the linked-list approach. However, for the `sup-sup` algorithm, the difference in total factorization time using the two approaches is negligible because the total time required to traverse the row-structure sets is extremely small in this algorithm for both techniques. Because overall factorization times differ by so little when the two approaches are compared, we have not included timing results for the more complicated of the two (the tree-traversal method) in §5. The important point to note is that either approach can be used in the `sup-sup` algorithm, and moreover, we believe that the tree-walking technique may ultimately be preferable in a parallel implementation for shared-memory multiprocessors [26].

4.3. Enhancements to the multifrontal method. The size of the stack of update matrices in the multifrontal method is a major issue associated with this method. A large stack obviously requires greater storage; perhaps not so obvious is that a large stack usually creates a great deal of overhead data movement that can erode efficiency. We have implemented two variants of the multifrontal algorithm. The first is a straightforward implementation of the algorithm in Fig. 7. One standard enhancement has been incorporated into both the basic and enhanced multifrontal code. Using a technique introduced by Liu [22], we have reordered the children of each parent in the supernodal elimination tree to minimize the storage requirement for the stack. This section describes the techniques incorporated solely into our enhanced version of the multifrontal method.

We are aware of multifrontal implementations [32] that compute the new factor columns $\boldsymbol{L}_{*,J}$ in factor storage rather than in $\boldsymbol{F}_J$ and then compute only the update matrix $\boldsymbol{U}_J$ within the frontal matrix $\boldsymbol{F}_J$. This simple change reduces the size of the frontal matrix and eliminates the need to move matrix columns back and forth between factor storage and the frontal matrix. We have implemented this technique and also further pursued the idea of reducing stack storage and limiting data movement by incorporating updates into factor storage as early as possible. More specifically, our enhanced version of the multifrontal method computes the update matrix $\boldsymbol{U}_J$ in working storage just as the basic version does. The enhanced version, however, incorporates some leading columns of $\boldsymbol{U}_J$ into factor storage, and thus stores on the stack only the trailing columns of $\boldsymbol{U}_J$ that were not incorporated into factor storage. It uses the following two techniques to implement this scheme in an efficient manner.

First, let P be the parent of J in the supernodal elimination tree. We say that J is *dense relative to P* if

$$Struct(\boldsymbol{L}_{*,P}) \subseteq Struct(\boldsymbol{L}_{*,J}).$$

If J is dense relative to P, then the update $cmod(P, J)$, which would normally fill the leading columns of $\boldsymbol{U}_J$, can be applied directly to $\boldsymbol{L}_{*,P}$, the columns of P in factor storage. This shrinks the size of the update matrix $\boldsymbol{U}_J$, and thus reduces data movement when $\boldsymbol{U}_J$ is ultimately moved to its final position at the top of the stack. Since this condition usually holds for the root supernode and one or more of its children, both of which usually have very large frontal matrices, this simple enhancement can save a lot of storage.

The second technique pushes this idea a bit further. Consider the update matrix $\boldsymbol{U}_J$ and again consider J's parent P. For the multifrontal method, the relative indices of each child with respect to its parent have been computed in advance. The relative indices used to assemble $\boldsymbol{U}_J$ into $\boldsymbol{F}_P$ can also be used to assemble the columns of $\boldsymbol{U}_J$ destined for $\boldsymbol{L}_{*,P}$ directly into factor storage. But there is no reason to limit this technique to the parent just because only the indices relative to P are available. If J happens to update its grandparent supernode P', then J's indices relative to P' can be obtained by gathering the appropriate indices of P (relative to P') into an integer work vector *relind*; they can then be used to assemble the appropriate columns of $\boldsymbol{U}_J$ into factor storage (i.e., $\boldsymbol{L}_{*,P'}$). If J happens to update its great-grandparent P'', then the process can be repeated with the old indices in *relind* (relative to P') used to gather some of the indices of P' (relative to P'') into *relind*, giving us the indices of J relative to P''.

The enhanced algorithm continues this process until it encounters either the last supernode requiring updates from the columns of J or an ancestor of J that is *not* updated by the columns of J. In consequence, factor storage for an ancestor supernode K of J is updated by the columns of J during this process if and only if factor storage for every ancestor on the path from J to K is similarly updated by the columns of J. When the first stopping criterion terminates the process, all columns of $\boldsymbol{U}_J$ have been incorporated into factor storage, and no columns of the update matrix $\boldsymbol{U}_J$ are placed on the stack. (One could think of it as stacking a null update matrix.) When the second stopping criterion terminates the process, the algorithm stacks a reduced update matrix, which includes only the trailing columns of $\boldsymbol{U}_J$ that have not been incorporated into factor storage. Thus, each assembly into factor storage reduces the amount of storage required for the reduced version of $\boldsymbol{U}_J$ and the amount of time required to move it to the top of the stack. The only overhead computation required, the sequence of integer gather operations, is negligible compared to the savings in data movement, and this technique is surprisingly effective at reducing the stack storage requirement, as we shall see in §5.

Last, one commonly used stack-reduction technique is the *in-place extension* of the update matrix for the child on top of the stack into the parent's new frontal matrix, which is initially set to zero. Liu [22] points out that this technique is used in the Harwell MA27 code, and Ashcraft [3] reports that overlapping the new frontal matrix with the topmost update matrix in this fashion saves a surprising 15%–27% in stack storage for his test problems. We have incorporated it into our enhanced multifrontal code.

4.4. Refinements for left-looking `sup-col` and `sup-sup` Cholesky factorization algorithms. Three refinements have been incorporated into our implementations of the left-looking `sup-col` and `sup-sup` Cholesky factorization algorithms, several of which concern the incorporation of update columns that are dense relative to the target column directly into factor storage. First, whenever K has only one column, the `sup-col` (`sup-sup`) code accumulates the column modification $cmod(j, K)$ ($cmod(J, K)$) directly into factor storage, avoiding use of the real work vector $\boldsymbol{t}$ ($\boldsymbol{T}$) altogether. This is extremely simple to implement, avoids some useless data movement, and is valuable for problems with many singleton supernodes. Second, all column modifications where the source and target columns come from the same supernode are performed as dense updates incorporated directly into factor storage using no indirect indexing. That is, they are performed as a dense update would be performed. Third, whenever the length of update columns from K matches the length of a target column(s) from J, it is also handled as a dense update. No indirect indexing is used, and the update is accumulated directly into factor storage.

Two minor refinements are incorporated into the left-looking `sup-sup` Cholesky algorithm only. Unlike the `sup-col` algorithm, the block algorithm explicitly computes and records relative indices as they are needed. By taking the difference between the first and the last of these indices and checking the difference against the length of the target, the algorithm is now capable of checking for *all remaining* dense updates, thereby avoiding some data movement and indirect addressing normally associated with these operations. Finally, note that the size of the block of working storage $\boldsymbol{T}$ needed by the algorithm is the size of the largest block update $cmod(J, K)$ generated by the algorithm that is *not dense* relative to its target factor columns. In practice, the children of the root supernode are usually dense relative to their parent and, moreover, the largest block update is often found among the block updates they generate for the root. Consequently, the practice of accumulating dense block updates directly into factor storage often reduces the amount of working storage needed for the algorithm.

5. Performance results. In this section we compare the performance of various sparse Cholesky factorization algorithms discussed in this paper, which include

- left-looking `col-col` Cholesky,
- left-looking `sup-col` Cholesky,
- left-looking `sup-sup` Cholesky,
- a basic multifrontal method, and
- an enhanced multifrontal method.

All algorithms were coded in Fortran, and all floating-point operations were performed in double precision, except on the Cray Y-MP. The code for left-looking `col-col` Cholesky was taken from SPARSPAK [8]. All codes were compiled with optimization turned on and were run on a vector supercomputer and a number of high-performance scientific workstations. It should be noted that identical code was run on each machine, except for the level of loop unrolling used in the block update routines.

The machines used in the experiments include

- an IBM RS/6000 model 530,
- a DEC 5000,
- a Stardent P3000, and
- one processor of a Cray Y-MP.

Each of the workstations has 64 kilobytes of cache memory. The cache on the IBM RS/6000 is four-way set-associative, while those on the DEC 5000 and Stardent P3000 are direct mapped. The cache line size on the IBM RS/6000 is 128 bytes, compared to 4 bytes on the DEC 5000 and Stardent P3000. The IBM RS/6000 and DEC 5000 have 16 megabytes of main memory, while the Stardent P3000 has 32 megabytes. Since we restricted our tests to problems that fit into the main memory, there was no paging and, hence, differences in memory size had no effect on performance. Both the DEC 5000 and Stardent P3000 use the same central processing unit (MIPS 3000), but they have different floating-point coprocessors. Moreover, the Stardent P3000 has special vector floating-point hardware that can be enabled or disabled during code compilation. The DEC 5000 and Stardent P3000 (with vectorization disabled) are similar in so many respects that we expect similar performance on these machines.

The vector supercomputer we used, the Cray Y-MP, has no memory hierarchy and has enough main memory for the largest of our test problems. It is also worth noting that this machine performs floating-point arithmetic far more efficiently than integer arithmetic, in contrast to the workstations where integer and floating-point performance is better balanced.

As we pointed out in previous sections, loop unrolling was employed in our implementation of the *cmod*(j, K) and *cmod*(J, K) block update operations. The optimal level of loop unrolling varies from machine to machine. In our experiments, we tried level-p loop unrolling, for $p = 1$, 2, 4, and 8. To limit the amount of data presented in our tables, we report data for only the level of loop unrolling that performed best on the specific machine under consideration. The best level was $p = 4$ for the DEC 5000 and Cray Y-MP, and $p = 8$ for the IBM RS/6000 and Stardent P3000.

Almost all the test problems were taken from the Harwell–Boeing Test Collection [13], which is widely used in testing and evaluating sparse matrix algorithms. The problems we selected and some of their characteristics are provided in Tables 1 and 2, respectively. To ensure that no paging occurs, only the small-to-medium-size problems were run on the workstations. All problems were run on the Cray Y-MP.

TABLE 1
List of test problems.

problem	brief description
BCSSTK13	Stiffness matrix – fluid flow generalized eigenvalues
BCSSTK14	Stiffness matrix – roof of Omni Coliseum, Atlanta
BCSSTK15	Stiffness matrix – module of an offshore platform
BCSSTK16	Stiffness matrix – Corp. of Engineers dam
BCSSTK17	Stiffness matrix – elevated pressure vessel
BCSSTK18	Stiffness matrix – R. E. Ginna nuclear power station
BCSSTK23	Stiffness matrix – portion of a 3D globally triangular bldg
BCSSTK24	Stiffness matrix – winter sports arena
BCSSTK25	Stiffness matrix – 76 story skyscraper
BCSSTK29	Stiffness matrix – buckling model of the 767 rear bulkhead
BCSSTK30	Stiffness matrix – offshore generator platform (MSC NASTRAN)
BCSSTK31	Stiffness matrix – automobile component (MSC NASTRAN)
BCSSTK32	Stiffness matrix – automobile chassis (MSC NASTRAN)
BCSSTK33	Stiffness matrix – pin boss (auto steering component), solid elements
NASA1824	Structure from NASA Langley, 1824 degrees of freedom
NASA2910	Structure from NASA Langley, 2910 degrees of freedom
NASA4704	Structure from NASA Langley, 4704 degrees of freedom

The tables presented in the following subsections contain the times required to run the factorization algorithms on several different machines. All execution times are in seconds. For machines that have cache memory, the notation `method(`s`)` is used, where `method` is either `sup-sup` or `mf` (multifrontal). When $s = 0$, supernodes are not subdivided into panels; when $s > 0$, large supernodes are subdivided into panels that fit into the s-kilobyte cache available on that machine. For example, on all the workstations $s = 64$ when the supernodes are subdivided. It is worth noting that all the test problems have many supernodes small enough to fit into cache, and both the multifrontal and left-looking `sup-sup` algorithms fully "reuse" the columns of such supernodes once they are loaded into cache, regardless of whether or not the larger supernodes have been subdivided to fit into cache.

5.1. IBM RS/6000. Table 3 contains the execution times (in seconds) required by the various factorization methods on an IBM RS/6000 model 530. We make the following observations from these results.

First, we see that `sup-col` consistently reduces factorization times by roughly a factor of 2 over `col-col`. Part of this large improvement is due to reductions in memory traffic and indirect addressing, which are, in turn, due respectively to the loop unrolling

TABLE 2
Characteristics of test problems.

problem	n	$\|A\|$	$\|L\|$	$\mu(L)$	N	flops
BCSSTK13	2,003	83,883	271,671	28,621	599	58,550,598
BCSSTK14	1,806	63,454	112,267	17,508	503	9,793,431
BCSSTK15	3,948	117,816	651,222	61,614	1,295	165,035,094
BCSSTK16	4,884	290,378	741,178	50,365	691	149,100,948
BCSSTK17	10,974	428,650	1,005,859	94,225	2,595	144,269,031
BCSSTK18	11,948	149,090	662,725	116,807	7,438	140,907,823
BCSSTK23	3,134	45,178	420,311	49,018	1,522	119,155,247
BCSSTK24	3,562	159,910	278,922	22,331	414	32,429,194
BCSSTK25	15,439	252,241	1,416,568	205,513	7,288	283,732,315
BCSSTK29	13,992	619,488	1,694,796	174,770	3,231	393,045,158
BCSSTK30	28,924	2,043,492	3,843,435	229,670	3,689	928,323,809
BCSSTK31	35,588	1,181,416	5,308,247	330,896	8,304	2,550,954,465
BCSSTK32	44,609	2,014,701	5,246,353	374,507	6,927	1,108,686,016
BCSSTK33	8,738	591,904	2,546,802	124,532	1,201	1,203,491,786
NASA1824	1,824	39,208	73,699	12,587	527	5,160,949
NASA2910	2,910	174,296	204,403	25,170	599	21,068,943
NASA4704	4,704	104,756	281,472	35,339	1,245	35,003,786

Legend:
n: number of equations,
$|A|$: number of nonzeros in A,
$|L|$: number of nonzeros in L, including the diagonal,
$\mu(L)$: number of row subscripts required to represent the supernodal structure of L,
N: number of fundamental supernodes in L,
flops: number of floating-point operations required to compute L.

TABLE 3
Factorization times in seconds on IBM RS/6000.

	`col-col`	`sup-col`	`sup-sup`		`basic mf`		`enhanced mf`	
problem			(0)	(64)	(0)	(64)	(0)	(64)
BCSSTK13	7.33	3.59	3.22	3.04	3.58	3.39	3.32	3.10
BCSSTK14	1.32	.69	.61	.61	.71	.69	.65	.65
BCSSTK15	20.40	9.68	8.77	8.08	9.49	8.78	8.98	8.32
BCSSTK16	18.61	8.94	7.93	7.47	8.59	8.15	8.01	7.52
BCSSTK18	17.86	9.30	8.58	8.07	9.39	9.08	8.99	8.47
BCSSTK23	14.71	7.13	6.57	6.00	7.21	6.67	6.78	6.26
BCSSTK24	4.28	2.03	1.76	1.72	1.92	1.88	1.78	1.74
NASA1824	.74	.41	.36	.36	.40	.40	.36	.36
NASA2910	2.81	1.46	1.24	1.23	1.36	1.36	1.26	1.25
NASA4704	4.56	2.29	2.01	1.94	2.17	2.10	2.03	1.96

and the dense matrix-vector multiplication used to implement the *cmod*(j, K) operation. However, the improvement of `sup-col` over `col-col` observed on this machine is considerably larger than that observed on the other workstations, which obtain the same reductions in memory traffic and indirect indexing. We believe that the large cache line size (128 bytes) on the IBM RS/6000 is responsible largely for this phenomenon. The memory-access pattern of the `col-col` algorithm is far more disordered and contains far fewer stride one vector reads and writes[3] than that of the `sup-col` algorithm. As a result, the `sup-col` algorithm is far more likely to use most or all of the floating-point numbers in a line as it is loaded into cache. Consequently, it often uses several (up to $16 = 128/8$) double precision numbers at the cost of a single cache miss.

[3]A stride one read or write accesses contiguous entries in a vector or array.

We see that `sup-sup(0)` usually improves performance over `sup-col` by 10%–15%. This improvement is partly due to further reductions in the cost of indirect indexing and the integer overhead associated with the row-structure lists. It is likely, however, that most of the improvement is due to reuse of data in the cache when the supernodes are small enough.

By partitioning supernodes whose columns overflow the cache into panels of contiguous columns that fit into cache and, moreover, by organizing the matrix–matrix multiplication operations to operate on these panels, data in cache is reused more effectively, and thus the amount of data moved to and from main memory is reduced. This leads to another 6%–10% improvement in factorization times for `sup-sup(64)` when it is used on the medium- and large-sized problems in the test set. Smaller increases are obtained for the small problems because most or all of their supernodes already fit into cache. This improvement is quite modest compared to that observed on the other workstations. We further explore this issue in the next subsection.

As expected, subdividing supernodes into panels that fit into cache improves the performance of both the basic and enhanced multifrontal methods in much the same manner that it improves the performance of the `sup-sup` algorithm. Enhanced `mf` performs significantly better than basic `mf`, probably because the former method typically requires much less data movement. Similarly, `sup-sup` performs slightly better than enhanced `mf`, probably because the former requires more data movement than the latter, despite the enhancements. Where applicable, these observations hold true on the other machines as well.

One of the most widely used implementations of the multifrontal method is the MA27 routine in the Harwell library [14]. To verify that our implementations of this method are adequate for fair comparisons, we have compared their performance with that of the MA27 routine in Table 4. Since loop unrolling and techniques for exploiting cache memory have not been incorporated into MA27, the fairest comparison is between the first two columns of the table. The second column contains the times required by basic `mf` with no loop unrolling and with no subdivision of the supernodes to improve cache usage. While this code outperforms MA27, the comparison is not really fair because of the additional cost of MA27's extremely flexible method for inputting the matrix entries. In any case, it is clear that our code is quite competitive. The last column demonstrates the value of the enhancements incorporated into our best implementation of the multifrontal method.

TABLE 4
Comparing 3 multifrontal methods: Factorization times in seconds on IBM RS/6000 (*basic* `mf` *does not use loop unrolling and enhanced* `mf` *uses level-*8 *loop unrolling*).

problem	MA27	basic `mf(0)` `level=1`	enhanced `mf(64)` `level=8`
BCSSTK13	5.88	4.98	3.10
BCSSTK14	1.31	.90	.65
BCSSTK15	15.38	13.44	8.32
BCSSTK16	15.02	12.22	7.52
BCSSTK18	14.95	12.76	8.47
BCSSTK23	11.17	10.15	6.26
BCSSTK24	3.74	2.65	1.74
NASA1824	0.74	.50	.36
NASA2910	2.88	1.81	1.25
NASA4704	3.83	2.96	1.96

TABLE 5
Factorization times in seconds on DEC 5000.

problem	col-col	sup-col	sup-sup (0)	sup-sup (64)	basic mf (0)	basic mf (64)	enhanced mf (0)	enhanced mf (64)
BCSSTK13	24.33	18.77	15.02	10.51	17.45	12.92	15.45	10.90
BCSSTK14	3.50	2.53	1.84	1.84	2.40	2.40	1.96	1.97
BCSSTK15	70.84	52.60	44.36	28.49	48.86	33.06	45.29	29.31
BCSSTK16	61.10	47.13	36.29	25.90	40.85	30.41	36.70	26.27
BCSSTK18	60.38	46.47	39.44	27.30	46.75	34.38	41.65	29.29
BCSSTK23	50.86	38.80	34.30	21.53	38.75	25.86	35.36	22.49
BCSSTK24	12.41	9.18	6.39	5.67	7.45	6.73	6.47	5.75
NASA1824	1.87	1.36	1.04	1.04	1.32	1.32	1.07	1.07
NASA2910	7.96	6.01	4.05	3.89	4.99	4.84	4.16	3.99
NASA4704	14.01	10.60	7.69	6.28	8.89	7.47	7.82	6.41

5.2. DEC 5000. Table 5 contains factorization times for the various factorization methods on a DEC 5000. In contrast to the IBM RS/6000, the reduction in factorization time of sup-col over col-col is only 30%–38%. (All percentages used in comparisons are relative to the smaller of the two times.) Due to the 4-byte cache line size on the DEC 5000, the contrasting memory access patterns of the col-col and sup-col algorithms do not incur, respectively, nearly as severe a penalty or nearly as great a performance boost as those noted earlier on the IBM workstation.

However, the improvement of the sup-sup algorithm over the sup-col algorithm is much more substantial on this machine. Mainly due to the 4-byte cache line and the larger penalty associated with each cache miss (two misses per floating-point number), the sup-sup algorithm generally obtains very significant performance improvements over the sup-col algorithm, whose capacity to reuse data in cache is quite limited for the larger test problems. The sup-sup(0) algorithm improves performance over the sup-col algorithm by 13%–30% for the larger problems and 31%–48% for the smaller problems. The sup-sup(64) algorithm improves performance over the sup-sup(0) algorithm by 40%–59% for the larger problems and 0%–22% for the smaller problems. The cumulative improvement is 70%–85% for the larger problems and 4%–48% for the smaller problems. A more detailed look at the the effect of cache size and organization on the performance of both the sup-sup and multifrontal algorithms can be found in Rothberg and Gupta [28].

5.3. Stardent P3000 (without vectorization). As mentioned earlier in this section, the Stardent P3000 and DEC 5000 have identical central processing units but different floating-point coprocessors. Thus, when vectorization is not used on the Stardent P3000, we expect performance to be quite similar on these two machines. The results in Tables 5 and 6 indicate that this is indeed the case, with one exception. For reasons we don't understand, loop unrolling is considerably less effective on this machine than it is on the DEC 5000. With the exception of the col-col to sup-col comparison, the various methods compare with each other very much as they did on the DEC 5000.

5.4. Stardent P3000 (with vectorization). The Stardent P3000 has floating-point vector hardware, which can be enabled or disabled when the code is compiled. Table 7 contains factorization times for the various factorization methods with *vectorization turned on*. An important observation is that subdividing the supernodes into panels that fit into the 64K cache has virtually no effect on performance. To avoid the

TABLE 6
Factorization times in seconds on Stardent P3000 (*without vectorization*).

problem	col-col	sup-col	sup-sup		basic mf		enhanced mf	
			(0)	(64)	(0)	(64)	(0)	(64)
BCSSTK13	28.75	24.80	19.09	12.62	20.97	14.43	19.49	12.96
BCSSTK14	3.82	3.17	2.15	2.15	2.53	2.54	2.28	2.29
BCSSTK15	84.70	69.33	56.50	34.20	60.08	37.64	57.42	35.00
BCSSTK16	72.25	62.84	45.67	31.09	49.11	34.45	46.26	31.33
BCSSTK18	72.00	61.75	50.09	32.83	55.50	37.96	52.18	34.58
BCSSTK23	60.66	51.08	44.01	25.85	47.65	29.34	45.12	26.81
BCSSTK24	14.11	11.84	7.71	6.70	8.40	7.39	7.74	6.73
NASA1824	2.03	1.69	1.21	1.21	1.39	1.39	1.23	1.24
NASA2910	8.94	7.64	4.82	4.58	5.44	5.21	4.86	4.64
NASA4704	16.34	13.87	9.45	7.46	10.25	8.26	9.54	7.55

TABLE 7
Factorization times in seconds on Stardent P3000 (*with vectorization*).

problem	col-col	sup-col	sup-sup		basic mf		enhanced mf	
			(0)	(64)	(0)	(64)	(0)	(64)
BCSSTK13	20.04	4.98	4.65	4.68	5.01	5.01	4.73	4.76
BCSSTK14	3.78	1.18	1.13	1.13	1.26	1.26	1.20	1.21
BCSSTK15	55.37	13.07	12.18	12.13	13.01	12.93	12.49	12.45
BCSSTK16	51.09	12.68	11.48	11.51	11.98	12.05	11.42	11.49
BCSSTK18	48.46	13.59	12.80	12.81	13.31	13.33	13.23	13.22
BCSSTK23	39.75	9.62	9.03	9.01	9.74	9.65	9.29	9.26
BCSSTK24	11.92	3.11	2.93	2.92	3.06	3.07	2.92	2.93
NASA1824	2.10	.74	.73	.73	.75	.75	.72	.71
NASA2910	7.90	2.33	2.20	2.21	2.31	2.32	2.20	2.21
NASA4704	12.76	3.58	3.36	3.37	3.49	3.49	3.36	3.37

complication of resolving cache misses during a vector operation, the vector hardware bypasses the cache altogether, and instead reads data directly from main memory in a pipelined fashion. This explains why paneling the supernodes is entirely ineffective in the sup-sup and the two multifrontal algorithms. It is worth noting, however, that reduced integer overhead and reduced indirect indexing in the sup-sup and multifrontal algorithms enable them to run faster than the sup-col algorithm. For instance, the sup-sup algorithm runs 5%–10% faster than the sup-col algorithm (excluding the two smallest problems from consideration). Evidently, our implementation of the dense matrix update kernels performs well on the Stardent P3000's vector hardware. For example, sup-sup is about 3.8–4.5 times faster than col-col (again, excluding the two smallest problems from consideration).

5.5. Cray Y-MP. Unlike the workstations considered in previous subsections, the Cray Y-MP has no cache memory. Its floating-point hardware is extremely fast due to vector pipelining. We have run the codes on a Cray Y-MP, and the results are provided in Table 8. As observed in [7] and [26], sup-col generally outperforms col-col by roughly a factor of 2. The use of loop unrolling, dense matrix-vector multiplication kernels, and the consequent large reductions in indirect addressing are responsible for these gains in performance. For medium to large problems, sup-sup outperforms sup-col by 6%-21%. (The performance gains are larger for the smaller problems.) The improvement is due to reductions in the cost of the indirect indexing and other integer processing overhead. The differences in performance among sup-sup, basic mf, and enhanced mf are very small.

TABLE 8
Factorization times in seconds on CRAY Y-MP.

problem	col-col	sup-col	sup-sup	basic mf	enhanced mf
BCSSTK13	0.84	0.42	0.36	0.38	0.38
BCSSTK14	0.22	0.15	0.11	0.12	0.12
BCSSTK15	2.22	1.02	0.90	0.96	0.95
BCSSTK16	2.18	1.02	0.89	0.92	0.89
BCSSTK17	2.46	1.29	1.07	1.11	1.09
BCSSTK18	2.05	1.23	1.14	1.15	1.22
BCSSTK23	1.56	0.75	0.67	0.72	0.72
BCSSTK24	0.62	0.32	0.26	0.27	0.26
BCSSTK25	4.20	2.42	2.13	2.13	2.20
BCSSTK29	5.45	2.79	2.38	2.47	2.48
BCSSTK30	12.73	5.52	4.96	5.12	4.99
BCSSTK31	28.96	12.16	11.48	11.58	11.43
BCSSTK32	15.98	7.38	6.47	6.63	6.49
BCSSTK33	13.68	5.61	5.29	5.47	5.30
NASA1824	0.14	0.11	0.09	0.08	0.09
NASA2910	0.43	0.27	0.21	0.21	0.21
NASA4704	0.65	0.40	0.33	0.32	0.33

5.6. Working storage requirements. The preceding subsections compare the *time* efficiency of the sparse Cholesky factorization algorithms under study on various high-performance uniprocessor computers. This subsection compares the *storage* efficiency of the various Cholesky factorization algorithms. More specifically, we computed the amount of auxiliary *floating-point* storage locations required by each method for accumulating column updates. Note that this ignores the floating-point storage required for the nonzero entries of L, which is the same for each method. It also ignores the amount of *integer* working storage required, since it is the sum of a small number of quantities $\leq n$, where n is the order of A, and, hence, does not vary much from one method to the next.

The `col-col` and `sup-col` algorithms have the lowest auxiliary working storage requirement because the columns are computed one at a time. For `col-col`, a floating-point work array of length n is needed to accumulate the updates. For the `sup-col` algorithm, the size of the floating-point work array is the maximum, over all columns $L_{*,j}$, of the number of nonzero entries in $L_{*,j}$. (Recall that an extra integer n-vector *indmap* is required to implement the indirect indexing scheme.)

The `sup-sup` and `mf` algorithms require more floating-point working storage. The two versions of the multifrontal method need auxiliary floating-point storage for stacking the update matrices. The `sup-sup` algorithm needs auxiliary floating-point storage to accumulate individual block updates $cmod(J, K)$. Thus, we are particularly interested in the storage requirements for `sup-sup` and `mf`.

Table 9 reports the floating-point working storage requirements for each method, normalized as a *percentage* of the number of nonzeros in L. As expected, the `sup-col` and `col-col` methods do indeed require the smallest amount of floating-point working storage. Without the enhancements to reduce the stack usage, the basic multifrontal method requires by far the most working storage. For two problems, the size of the stack is roughly 60% of the "size" of L. The enhanced multifrontal algorithm required far less floating-point working storage than the basic multifrontal algorithm requires, but still considerably more than the `sup-sup` algorithm requires.

6. Concluding remarks. We have studied three different left-looking sparse Cholesky factorization algorithms: the `col-col`, `sup-col`, and `sup-sup` algorithms. The use

TABLE 9
Floating-point working storage (percent of $|L|$).

problem	col-col	sup-col	sup-sup	basic mf	enhanced mf
BCSSTK13	.74	.14	8.63	58.70	15.80
BCSSTK14	1.61	.16	4.08	27.49	6.79
BCSSTK15	.61	.07	5.91	30.44	7.79
BCSSTK16	.66	.04	2.63	17.12	3.90
BCSSTK17	1.09	.03	1.90	12.43	2.70
BCSSTK18	1.80	.06	5.06	25.37	7.46
BCSSTK23	.75	.12	12.18	60.07	17.31
BCSSTK24	1.28	.09	4.80	23.45	5.46
BCSSTK25	1.09	.03	2.34	11.81	3.27
BCSSTK29	.83	.03	3.05	17.01	7.16
BCSSTK30	.75	.02	1.69	11.22	2.46
BCSSTK31	.67	.02	3.27	24.21	5.49
BCSSTK32	.85	.01	1.10	8.50	2.02
BCSSTK33	.34	.04	5.15	40.31	9.06
NASA1824	2.47	.22	4.51	39.66	8.59
NASA2910	1.42	.10	3.96	25.93	5.99
NASA4704	1.67	.10	6.81	29.16	8.72

of supernode-to-column updates in the `sup-col` algorithm (instead of the column-to-column updates in the `col-col` algorithm) reduces the amount of memory traffic and indirect addressing overhead. Our tests have shown the effectiveness of this well known technique on a wide range of high-performance uniprocessor computers. The use of supernode-to-supernode updates in the `sup-sup` algorithm further reduces the amount of memory traffic on machines with high-speed local memory, such as a cache. For our test problems, the `sup-sup` algorithm obtains virtually the same performance improvements via reuse of cached data that the multifrontal method obtains. Similar test results have appeared in Rothberg and Gupta [28]. On machines without a cache, the `sup-sup` algorithm obtains modest improvements over the `sup-col` algorithm by further reducing the integer overhead and indirect indexing costs.

Although the performance of the various left-looking factorization algorithms is machine dependent, it is interesting to note that for three high-performance workstations (the IBM RS/6000, the DEC 5000, and the Stardent P3000 without vectorization), the `sup-sup` algorithm with subdivided supernodes is the most efficient algorithm and often runs 2.5 times faster than the `col-col` algorithm. On the Stardent P3000 with vectorization, the `sup-sup` algorithm is roughly 4–4.5 times faster than the `col-col` algorithm.

For the test problems and workstations considered in this paper, the enhanced multifrontal algorithm is slightly slower than the `sup-sup` algorithm (by roughly 5%–10%). The results also indicate that the enhancements we have made to the multifrontal method greatly reduce the amount of auxiliary storage required for the stack and the amount of data movement required to stack the update matrices. The working storage requirement in `sup-sup`, however, remains smaller than that in the enhanced multifrontal method.

One of the goals in this study is to identify the "best" sequential sparse Cholesky factorization algorithm. This algorithm will be used to evaluate the performance of various parallel sparse Cholesky factorization methods. Based on our results, we conclude that the left-looking `sup-sup` algorithm is the most efficient algorithm, both in terms of its execution time and working storage requirement. Parallel versions of left-looking `col-col` and `sup-col` algorithms have appeared in [17] and [26], respectively. Parallel implementation of the left-looking `sup-sup` algorithm is currently under investigation, and performance results will be reported elsewhere.

It should be noted that the basic multifrontal method has at least two advantages over the left-looking methods. First, the multifrontal algorithm has long been recognized as the better candidate for out-of-core implementation: Only the stack of update matrices and the current frontal matrix are needed in main memory. Second, its superior data locality is of great value when solving very large problems on machines with virtual memory and paging [23]. The impact of paging on performance is not considered in this paper because our main concern is the working storage requirement and the use of blocking to exploit the first level in the memory hierarchy (i.e., fast memory or cache). The paging issue, however, will be investigated elsewhere.

Acknowledgments. The authors thank the Minnesota Supercomputer Institute and Cray Research, Inc. for providing computer support. Also, part of the work was performed while the authors were visiting the Institute for Mathematics and its Applications at the University of Minnesota, which is funded principally by the National Science Foundation.

REFERENCES

[1] P. Amestoy and I. Duff, *Vectorization of a multiprocessor multifrontal code*, Internat. J. Supercomput. Appl., 3 (1989), pp. 41–59.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK: A portable linear algebra library for high-performance computers*, in Proceedings of Supercomputing '90, IEEE Press, 1990, pp. 1–10.

[3] C. Ashcraft, *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Tech. Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, WA, 1987.

[4] ———, *The domain/segment partition for the factorization of sparse symmetric positive matrices*, Tech. Report ECA-TR-148, Engineering Computing and Analysis Division, Boeing Computer Services, Seattle, WA, 1990.

[5] ———, Personal communication, 1991.

[6] C. Ashcraft, S. Eisenstat, J. Liu, and A. Sherman, *A comparison of three column-based distributed sparse factorization schemes*, Tech. Report YALEU/DCS/RR-810, Department of Computer Science, Yale University, New Haven, CT, 1990.

[7] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Internat. J. Supercomput. Appl., 1 (1987), pp. 10–30.

[8] E. Chu, A. George, J. W.-H. Liu, and E. G.-Y. Ng, *User's guide for SPARSPAK-A: Waterloo sparse linear equations package*, Tech. Report CS-84-36, University of Waterloo, Waterloo, Ontario, Canada, 1984.

[9] A. Dave and I. Duff, *Sparse matrix calculations on the Cray-2*, Parallel Comput., 5 (1987), pp. 55–64.

[10] J. Dongarra and S. Eisenstat, *Squeezing the most out of an algorithm in Cray Fortran*, ACM Trans. Math. Software, 10 (1984), pp. 219–230.

[11] J. Dongarra, F. Gustavson, and A. Karp, *Implementing linear algebra algorithms for dense matrices on a vector pipeline machine*, SIAM Rev., 26 (1984), pp. 91–112.

[12] I. Duff, A. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, England, 1987.

[13] I. Duff, R. Grimes, and J. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[14] I. Duff and J. Reid, *MA27—A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Tech. Report AERE R 10533, Harwell, England, 1982.

[15] ———, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[16] S. Eisenstat, M. Gursky, M. Schultz, and A. H. Sherman, *The Yale sparse matrix package I. the symmetric codes*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1145–1151.

[17] A. George, M. Heath, J. W.-H. Liu, and E. G.-Y. Ng, *Solution of sparse positive definite systems on a shared memory multiprocessor*, Internat. J. Parallel Programming, 15 (1986), pp. 309–325.

[18] A. George, M. Heath, J. W.-H. Liu, and E. G.-Y. Ng, *Sparse Cholesky factorization on a local-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 327–340.

[19] A. George and J. W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[20] L. Hulbert and E. Zmijewski, *Limiting communication in parallel sparse Cholesky factorization*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1184–1197.

[21] J. W.-H. Liu, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. Math. Software, 12 (1986), pp. 127–148.

[22] ———, *On the storage requirement in the out-of-core multi-frontal method for sparse factorization*, ACM Trans. Math. Software, 12 (1986), pp. 249–264.

[23] ———, *The multifrontal method and paging in sparse Cholesky factorization*, ACM Trans. Math. Software, 15 (1989), pp. 310–325.

[24] ———, *The multifrontal method for sparse matrix solution: theory and practice*, SIAM Rev., 34 (1992), pp. 82–109.

[25] ———, *A generalized envelope method for sparse factorization by rows*, ACM Trans. Math. Software, 17 (1991), pp. 112–129.

[26] E. G. Ng and B. Peyton, *A supernodal Cholesky factorization algorithm for shared-memory multiprocessors*, SIAM J. Sci. Comput., 14 (1993), pp. 761–769.

[27] E. Rothberg and A. Gupta, *A comparative evaluation of nodal and supernodal parallel sparse matrix factorization: Detailed simulation results*, Tech. Report STAN-CS-90-1305, Stanford University, Stanford, CA, 1990.

[28] ———, *An evaluation of left-looking, right-looking, and multifrontal approaches to sparse Cholesky factorization on hierarchical-memory machines*, Tech. Report STAN-CS-91-1377, Stanford University, Stanford, CA, 1991.

[29] ———, *Fast sparse matrix factorization on modern workstations—exploiting the memory hierarchy*, ACM Trans. Math. Software, 17 (1991), pp. 313–334.

[30] R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.

[31] A. Sherman, *On the efficient solution of sparse systems of linear and nonlinear equations*, Tech. Report 46, Department of Computer Science, Yale University, New Haven, CT, 1975.

[32] C. Yang, *A vector/parallel implementation of the multifrontal method for sparse symmetric definite linear systems on the Cray Y-MP*, Cray Research Inc., Mendota Heights, MN, 1990, manuscript.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1057–1071, September 1993

A MONOTONE PETROV–GALERKIN METHOD FOR QUASILINEAR PARABOLIC DIFFERENTIAL EQUATIONS*

ØISTEIN BØE[†]

Abstract. To stabilize convection-dominated flows, artificial diffusion is often introduced by upstream weighting. A fundamental question is therefore: What amount of artificial diffusion should be introduced to avoid numerical oscillations? Obviously, that amount needs to be minimized, since artificial diffusion causes smearing of sharp fronts and less spatial accuracy. This paper analyzes the Petrov–Galerkin method with linear basis functions and quadratic, asymmetric test functions. The author uses the ideas of monotonicity to control the artificial diffusion introduced so that the numerical schemes are a priori monotone, i.e., numerical oscillations will not form if the discrete equations are solved exactly.

Key words. Petrov–Galerkin method, monotonicity, monotonic matrices, M-matrices

AMS subject classifications. 65M60, 65M12, 76S06, 35K55

1. Introduction.

In this paper we study numerical solutions of the parabolic equation

$$\frac{\partial S}{\partial t} + \frac{\partial f}{\partial x} = 0, \qquad x \times t \in \Omega = (0,1) \times (0,\infty), \tag{1.1}$$

$$f(S) = A(S) + H(S) + D(S)\frac{\partial S}{\partial x}. \tag{1.2}$$

The convective flux consists of two parts, $A(S)$ and $H(S)$. $A(S)$ is a strictly increasing S-shaped function with $A(0) = 0$. $H(S)$ is a bell-shaped positive function of S with $H(0) = 0$ and $H(1) = 0$. The diffusive coefficient $D(S)$ is a negative function of S with possible zeros for $S = 0$ and $S = 1$.

We shall first state a monotonicity property for (1.1). Let $\Gamma = ([0,1] \times 0) \cup (0 \times [0,\infty)) \cup (1 \times [0,\infty))$ be the boundary of Ω. Then let S and S^* be two solutions of (1.1) in $\Omega \cup \Gamma$. Then the following continuous monotonicity property holds [22], [10]:

$$\text{If } S - S^* \geq 0 \text{ on } \Gamma \text{ then } S - S^* \geq 0 \text{ in } \Omega.$$

Equation (1.1), together with appropriate initial and boundary conditions, may model immiscible two-phase incompressible porous medium flow (such as oil and water) in a one-dimensional reservoir or core. S is then the water saturation. $A(S)$ represents the part of convection proportional to the total Darcy velocity u, while $H(S)$ represents the part proportional to the gravitational acceleration. $D(S)\partial S/\partial x$ represents diffusion caused by capillary pressure (difference in phase pressures) [20]. In industrial mathematics it is important to have structurally stable numerical methods at hand. The computer programs are used over and over for different sets of parameters. In this nonlinear application, numerical oscillations may force us to evaluate the flux function f for unphysical values of S. This may create unstable behavior, especially near the boundaries [4]. Consequently, monotonicity is an important property. In this paper, we will construct a Petrov–Galerkin method that has proved to give nonoscillatory numerical solutions. The method is capable of handling both Neumann and Dirichlet boundary conditions and variable mesh spacing.

*Received by the editors September 18, 1991; accepted for publication (in revised form) September 9, 1992.

[†]Norsk Hydro A/S, Research Centre, N-5020 Bergen, Norway.

Time t is discretized into time levels $t_n = t_{n-1} + s_n, n = 1, 2, \ldots$ with $t_0 = 0$. Without loss of generality for the analysis to come, we set $s_n = s$ for all n. The space interval $x \in [0, 1]$ is partitioned into N intervals defined by $N + 1$ points, $x_i = x_{i-1} + h_{i-1}, i = 1, 2, \ldots, N$ with $x_0 = 0$ and $x_N = 1$. Suppose some numerical scheme results in a discrete solution vector S_h^n at time level n. The components of S_h^n are $\{S_i^n\}, i = 0, 1, \ldots, N$, where each S_i^n represents the solution at the point given by the coordinate x_i. Moreover, let $S_h^n = F(S_h^{n-1})$ denote a two-level in-time explicit scheme, and let $G(S_h^n) = S_h^{n-1}$ denote a two-level in-time implicit scheme. F and G are vector functions having the same dimension as the vector S_h^n. It will be crucial for monotonicity properties of a numerical scheme that the discretization of the second-order term is based on the use of the following potential [1]:

$$\Psi(S) = \int_{S_r}^{S} D(S^*)dS^*, \tag{1.3}$$

where S_r is some reference value. $\Psi(S)$ will be referred to as the PSI potential.

We have $\partial\Psi/\partial x = D(S)\partial S/\partial x$ by the chain rule. $\Psi(S)$ is a decreasing function of S and might be tabulated by using the numerical integration of (1.3). Typical shapes of the nonlinear functions A, H, D, and Ψ are shown in Fig. 1.

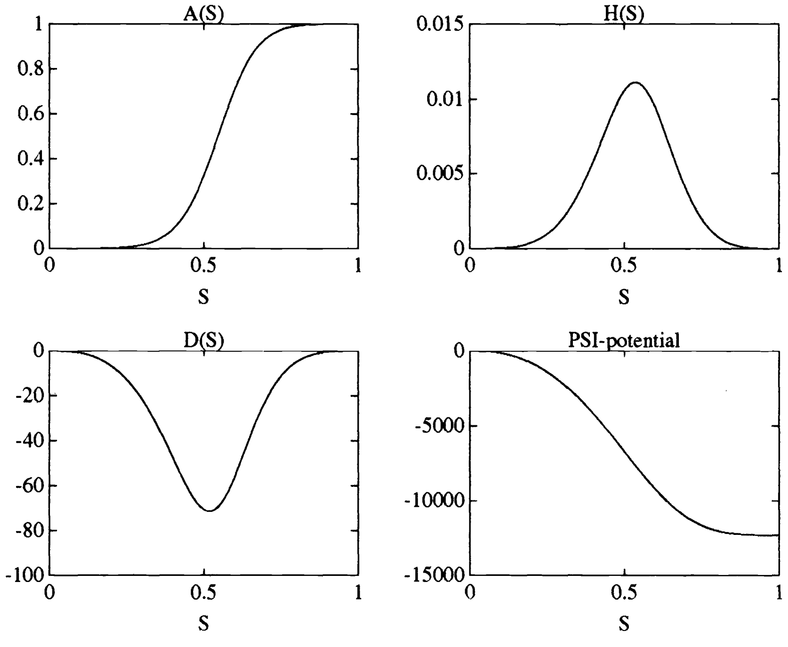


FIG. 1. *Nonlinear coefficients.*

Methods for avoiding or reducing numerical oscillations have been the theme of many papers. Much of this research has been concerned with hyperbolic conservation laws. Harten [12] introduced the concept of the TVD (total variation diminishing) property as a discrete version of the nonincreasing variation property of the solution of the continuous problem. The TVD property forces a condition on the coefficients of the numerical scheme. These "coefficients" are generally functions of both the discrete solution and its derivative. Many authors have used flux limiters as a tool for deriving TVD schemes. Such schemes represent a modification of a first-order scheme in the sense that they add a limited amount of antidiffusive flux in order to satisfy the TVD property. Sweby [25] compared various limiters both analytically and numerically. For hyperbolic conservation laws with variable mesh spacing, Sanders [24] showed convergence in L_1 for both implicit and explicit monotone difference schemes.

In this paper, we will formulate a discrete version of the continuous monotonicity property by using the concepts of M-matrices and monotone matrices. These concepts will be useful for analyzing the monotonicity properties of a numerical scheme. This analysis is based on examining the Jacobian matrix for the discrete system of difference equations. Consequently, a discrete equation centered at a certain node will be represented by a row in the Jacobian. The first and last rows will correspond to the discrete equation at the boundaries. Therefore the effect of boundary conditions is taken into account. The Petrov–Galerkin method will be constructed using these concepts.

The Petrov–Galerkin method is used for damping the numerical oscillations in the Galerkin method by adding numerical diffusion. Numerical oscillations may still form, although they appear to be more damped than for the Galerkin method. Petrov–Galerkin methods for diffusion-convection problems have been studied by several authors. It is well known that the use of linear basis functions and quadratic test functions will add numerical diffusion to the Galerkin method [19]. Barrett and Morton [2] introduced so-called optimal test functions in order to achieve an approximate symmetrization of the bilinear form arising from the variational formulation. Let H^1 be the Sobolev space where the functions and their first-order partial derivatives are square integrable. Let $B_m(\cdot,\cdot)$ be a symmetric coercive bilinear form on $H^1 \times H^1$. From the Riesz representation theorem there exists a representer $R_m : H^1 \times H^1$ such that $B(v,w) = B_m(v, R_m w)$ for all $v, w \in H^1$. The idea is to construct a test space T_h from the space of basis functions N_h such that $\text{span}(R_m T_i) = \text{span}(N_i)$ for each function i. This symmetrization (or approximate symmetrization in practice) will ensure that the optimal approximation property of the Galerkin method also holds (or "nearly" holds) for the Petrov–Galerkin method. Barrett and Morton applied their theory to constant and variable coefficient problems. The ideas of Barrett and Morton were adopted by Demkowicz and Oden [8] who presented a method for localizing the optimal test functions and obtained superconvergence results. Dahle, Espedal, and Ewing [7] used this formalism to obtain a Petrov–Galerkin method for (1.1)–(1.2) with Dirichlet boundary conditions.

The outline of this paper is as follows. In §2 we define monotonicity for implicit and explicit numerical schemes. Then we derive sufficient conditions for monotonicity by using the theory of M-matrices and monotone matrices. In §3 it is shown that monotonicity is not guaranteed for finite-element methods unless the mass matrix is lumped. Section 4 is concerned with the construction of the a priori monotone Petrov–Galerkin method. In §5 we present some numerical examples. Finally, we give some closing remarks in §6.

2. Monotonicity for implicit and explicit numerical schemes. We want to construct numerical schemes that possess a monotonicity property. The following definitions of monotonicity for numerical schemes approximating (1.1) seem natural in view of the "continuous" monotonicity property; see [1].

DEFINITION 1. An explicit numerical scheme $S_h^n = F(S_h^{n-1})$ is said to be monotone with respect to time if $S_h - S_h^* \geq 0$ implies $F(S_h) - F(S_h^*) \geq 0$.

DEFINITION 2. An implicit numerical scheme $G(S_h^n) = S_h^{n-1}$ is said to be monotone with respect to time if $G(S_h) - G(S_h^*) \geq 0$ implies $S_h - S_h^* \geq 0$.

In these two definitions the inequalities hold for all elements of the vectors. The essence of the two definitions above is as follows. Assume that we want to advance the solution from time level $n-1$ to n and assume $S_h^{n-1} - S_h^{n-2} \geq 0$. For the explicit schemes we have $S_h^n - S_h^{n-1} = F(S_h^{n-1}) - F(S_h^{n-2})$. Therefore, if $S_h^{n-1} - S_h^{n-2} \geq 0$, we would like to have $F(S_h^{n-1}) - F(S_h^{n-2}) \geq 0$ so that $S_h^n - S_h^{n-1} \geq 0$. For the implicit schemes we have $S_h^{n-1} - S_h^{n-2} = G(S_h^n) - G(S_h^{n-1})$. For the solution to be monotone in time, $G(S_h^n) - G(S_h^{n-1}) \geq 0$ must imply $S_h^n - S_h^{n-1} \geq 0$.

The above conditions for monotonicity in time could be analyzed by the concepts of monotonic matrices and M-matrices. For these concepts we refer to [3], [6], and [21]. Let A be a regular quadratic matrix with elements $\{a_{ij}\}$ of dimension $N+1$, and let y be a column vector of dimension $N+1$. If all the elements of A are nonnegative, we say that A is nonnegative and write $A \geq 0$, and if all components of the vector y are nonnegative, we write $y \geq 0$.

DEFINITION 3. The matrix A is monotone if $y \geq 0 \Rightarrow Ay \geq 0$. The implication is equivalent to $A \geq 0$.

DEFINITION 4. A matrix A is of monotone type if the implication $Ay \geq 0 \Rightarrow y \geq 0$ holds. The implication is equivalent to $A^{-1} \geq 0$.

DEFINITION 5. A matrix A is an L-matrix if the splitting $A = B - C$, where $B = \operatorname{diag}(b_i)$ is the diagonal matrix $b_i = a_{ii}$, has the property $B \geq 0$ and $C \geq 0$.

DEFINITION 6. A matrix A is an M-matrix if and only if the following conditions hold: (i) A is an L-matrix.

(ii) There exists a diagonal matrix $D = \operatorname{diag}(d_i) \geq 0$ such that DA is columnwise strictly diagonally dominant ($\sum_i a_{ij} d_i > 0$) or AD is rowwise strictly diagonally dominant ($\sum_j a_{ij} d_j > 0$).

Let A be an M-matrix with the splitting $A = B - C$ given in Definition 5. A^{-1} may be written

$$A^{-1} = B^{-1}(I - CB^{-1})^{-1}. \tag{2.1}$$

For the spectral radius $\rho(CB^{-1})$ we have

$$\begin{aligned} \rho(CB^{-1}) &= \rho(D(CB^{-1})D^{-1}) = \rho(DC(DB)^{-1}) \\ &\leq \|DC(DB)^{-1}\|_1 = \max_j \sum_i \frac{c_{ij} d_i}{b_j d_j} < 1. \end{aligned} \tag{2.2}$$

By use of the Neumann expansion [23] for the inverse of the matrix A, we get

$$A^{-1} = B^{-1}(I - CB^{-1})^{-1} = B^{-1} \sum_{k=0}^{\infty} (CB^{-1})^k \geq 0. \tag{2.3}$$

Consequently, an M-matrix is of monotone type.

Now assume that the vector functions F and G have a Gâteaux-derivative at each point of the interval $[S_h^*, S_h^{**}]$. Assume also that the derivatives of F and G are Riemann-integrable. Then the mean-value theorem holds in integral form [23, p. 70]. Let J denote the vector function F or G

$$\begin{aligned} J(S_h^{**}) - J(S_h^*) &= (S_h^{**} - S_h^*) \int_{S_h^*}^{S_h^{**}} [J'(S_h^* + t(S_h^{**} - S_h^*))]\, dt \\ &= E(S_h^{**} - S_h^*). \end{aligned} \tag{2.4}$$

Let the Jacobian J' be monotone (explicit case) or an M-matrix (implicit case). In both cases the integrated matrix E will have the same properties.

Now assume that $S_h^{**} - S_h^* \geq 0$ and $F'(S_h) \geq 0$ for all $S_h \in [S_h^*, S_h^{**}]$, then $F(S_h^{**}) - F(S_h^*) \geq 0$. Moreover, given $G(S_h^{**}) - G(S_h^*) \geq 0$ and the condition $G'(S_h)^{-1} \geq 0$ for all $S_h \in [S_h^*, S_h^{**}]$, then $S_h^{**} - S_h^* \geq 0$. Consequently, the explicit schemes are monotone in time if the Jacobian $F'(S_h)$ is monotone, and the implicit schemes are monotone if the Jacobian $G'(S_h)$ is an M-matrix (and hence of monotone type). We remark that these conditions are sufficient but not necessary for the solution to be monotone. It is an open question whether or not the conditions are too restrictive.

Using the above concepts it may be shown that one-point upstream weighting is monotone, while central differencing is not monotone. This will be clear from the analysis in §4.

Aavatsmark [1] discussed implicit finite-difference approximations and their monotonicity properties of (1.1). The implicit Lax–Friedrichs scheme [18] is monotone under the restrictive CFL (Courant–Friedrichs–Lewy) condition $s/h \leq 1/\|f\|_\infty$. The implicit variant of the first-order Godunov method [11] is unconditionally monotone, but the flux function $A(S)$ is not continuously differentiable. This may cause problems if Newton iteration is used. The Engquist–Osher method [9] is unconditionally monotone, and the flux function is differentiable. But the Engquist–Osher method introduces more artificial diffusion compared to the first-order Godunov method.

3. Generalization to finite-element schemes. For finite-element techniques, the mass matrix in front of the time derivatives generally contains nondiagonal contributions. Such schemes will be in the form

$$TS_h^n = F^*(S_h^{n-1}) + TS_h^{n-1}, \tag{3.1}$$

$$TS_h^{n-1} = G^*(S_h^n) + TS_h^n, \tag{3.2}$$

where T is the mass matrix.

To obtain such schemes in the standard form given in Definitions 1 and 2, multiply (3.1) and (3.2) by T^{-1} (assuming that T is nonsingular). The schemes then become

$$S_h = T^{-1}J^*(S_h^*) + S_h^* = J(S_h^*). \tag{3.3}$$

Here $J^* = F^*, S_h^* = S_h^{n-1}$, and $S_h = S_h^n$ for the explicit schemes and $J^* = G^*, S_h^* = S_h^n$, and $S_h = S_h^{n-1}$ for the implicit schemes. The Jacobian J' becomes

$$J' = I + T^{-1}J^{*\prime}. \tag{3.4}$$

The inversion of the matrix T complicates the monotonicity analysis. For the Petrov–Galerkin method presented in §4, the matrix T is tridiagonal and nonsymmetric. The inverse is generally a full matrix. We may avoid this problem by writing

$$J' = I + T^{-1}J^{*\prime} = T^{-1}(J^{*\prime} + T). \tag{3.5}$$

The inverse becomes

$$J'^{-1} = (T^{-1}(J^{*\prime} + T))^{-1} = (J^{*\prime} + T)^{-1}T. \tag{3.6}$$

According to (3.4), the explicit schemes are monotone in time if the matrix $T^{-1}J^{*\prime}$ is monotone. This condition may be met by requiring T to be an M-matrix and the Jacobian $J^{*\prime}$ to be nonnegative. For the implicit schemes we consider (3.6): It is clear that implicit schemes are monotone in time if the matrix T is monotone and if, in addition, the Jacobian $J^{*\prime} + T$ is of monotone type. This condition is satisfied if T is nonnegative and $J^{*\prime} + T$ is an M-matrix. A nondiagonal, nonnegative matrix T imposes a lower bound on the time step in order to ensure that $J^{*\prime} + T$ is an M-matrix. This is because T contains nonnegative elements of order $O(h/s)$ outside the main diagonal. In fact, this condition is also necessary in some cases; see §5 Example 1. This indicates that the mass matrix T should be lumped (diagonalized) in order to guarantee monotonicity.

4. An a priori monotone time-discretized Petrov–Galerkin method.

To simplify, we transform element i (the interval $x \in [x_i, x_{i+1}]$) to the unit element by

$$\xi_i = \frac{x - x_i}{x_{i+1} - x_i}, \quad i = 0, 1, \ldots, N-1. \tag{4.1}$$

Let $H^1(\langle 0,1\rangle)$ denote the Sobolev space on $\langle 0,1\rangle$ where the functions and their first-order derivatives are square integrable (i.e., members of L_2). Let (u, v) denote the inner product in L_2.

We now seek the discrete solution S_h in the finite dimensional subspace $M^1 \subset H^1$, consisting of piecewise linears.

On each element i the saturation is interpolated linearly by

$$\begin{aligned} S_{h_i} &= S_{i,1}v_{i,1}(\xi) + S_{i,2}v_{i,2}(\xi) \\ &= S_i v_{i,1}(\xi) + S_{i+1}v_{i,2}(\xi), \end{aligned} \tag{4.2}$$

with basis

$$v_{i,1}(\xi) = 1 - \xi, \tag{4.3}$$

$$v_{i,2}(\xi) = \xi, \tag{4.4}$$

where $S_{i,1} = S_i$ and $S_{i,2} = S_{i+1}$ are the unknowns at x_i and x_{i+1}, respectively.

For the test space we choose the asymmetric basis

$$T_{i,1}(\xi) = v_{i,1}(\xi) + 3\alpha_i\xi(1-\xi), \tag{4.5}$$

$$T_{i,2}(\xi) = v_{i,2}(\xi) - 3\alpha_i\xi(1-\xi), \tag{4.6}$$

where α_i is to be chosen to introduce artificial diffusion.

On each element, the interpolant S_{h_i} is substituted for S and the saturation equation is multiplied by the test functions (4.5)–(4.6) and integrated over the element

$$\begin{aligned} e_{i,j} = \left(h_i\left(\frac{\partial S_{h_i}}{\partial t}, T_{i,j}\right) - A(S_{h_i}) + \frac{1}{h_i}\frac{\partial \Psi(S_{h_i})}{\partial \xi}, \frac{\partial T_{i,j}}{\partial \xi}\right) \\ + fT_{i,j}\big|_0^1 = 0, \qquad j = 1, 2, \\ i = 0, 1, \ldots, N-1. \end{aligned} \tag{4.7}$$

The flux term is integrated by parts and the two-point trapezoidal rule is used. This will diagonalize the mass matrix in front of the time derivatives and, as we commented in §3, is necessary to guarantee monotonicity. By the discretization $\partial\Psi(S_{h_i})/\partial\xi = \Psi(S_{i,2}) - \Psi(S_{i,1})$, the quadratic terms of the test functions (4.5)–(4.6) will not affect the diffusion term in (1.1). We neglect gravity effects and define $H(S) \equiv 0$. The Petrov–Galerkin scheme is now constructed by forming the discrete system of equations. We set $E_0 = e_{0,1}$, $E_i = e_{i-1,2} + e_{i,1}$, $i = 1, 2, \ldots, N-1$, and $E_N = e_{N-1,2}$. In the summations for constructing E_i, the flux f will cancel over the interior element boundaries (conservation).

The following scheme results:

$$
\begin{aligned}
E_0 = {} & \frac{1}{2}h_0\frac{S_0^n - S_0^{n-1}}{s} \\
& + \frac{1}{2}\left((1-3\alpha_0)A(S_0^\nu) + (1+3\alpha_0)A(S_1^\nu)\right) \\
& + \frac{\Psi(S_1^\nu) - \Psi(S_0^\nu)}{h_0} + f|_{x=0} = 0,
\end{aligned}
\tag{4.8}
$$

$$
\begin{aligned}
E_i = {} & \frac{1}{2}(h_{i-1} + h_i)\frac{S_i^n - S_i^{n-1}}{s} \\
& + \frac{1}{2}\left(-(1-3\alpha_{i-1})A(S_{i-1}^\nu) - (3\alpha_{i-1} + 3\alpha_i)A(S_i^\nu) + (1+3\alpha_i)A(S_{i+1}^\nu)\right) \\
& - \frac{\Psi(S_i^\nu) - \Psi(S_{i-1}^\nu)}{h_{i-1}} + \frac{\Psi(S_{i+1}^\nu) - \Psi(S_i^\nu)}{h_i} = 0, \qquad i = 1, 2, \ldots, N-1,
\end{aligned}
\tag{4.9}
$$

$$
\begin{aligned}
E_N = {} & \frac{1}{2}h_N\frac{S_N^n - S_N^{n-1}}{s} \\
& - \frac{1}{2}\left((1-3\alpha_{N-1})A(S_{N-1}^\nu) + (1+3\alpha_{N-1})A(S_N^\nu)\right) \\
& - \frac{\Psi(S_N^\nu) - \Psi(S_{N-1}^\nu)}{h_{N-1}} - f|_{x=1} = 0.
\end{aligned}
\tag{4.10}
$$

The system given by (4.8)–(4.10) is written in the form $S_h^n = F(S_h^{n-1})$ (explicit case, $\nu = n-1$) and $S_h^{n-1} = G(S_h^n)$ (implicit case, $\nu = n$).

Let

$$
\chi_i = \begin{cases} h_0 & \text{if } i = 0, \\ h_{i-1} + h_i & \text{if } 1 \le i \le N-2, \\ h_{N-1} & \text{if } i = N-1. \end{cases}
\tag{4.11}
$$

The column structures of the tridiagonal Jacobians F' and G' become

$$
\frac{\partial F_0}{\partial S_0} = 1 - (1-3\alpha_0)\frac{sA'(S_0)}{\chi_0} + 2\frac{s\Psi'(S_0)}{h_0\chi_0},
\tag{4.12}
$$

$$\frac{\partial F_{i-1}}{\partial S_i} = -(1+3\alpha_{i-1})\frac{sA'(S_i)}{\chi_{i-1}} - 2\frac{s\Psi'(S_i)}{h_{i-1}\chi_{i-1}}, \qquad i=1,2,\ldots,N, \tag{4.13}$$

$$\frac{\partial F_i}{\partial S_i} = 1 + (3\alpha_{i-1}+3\alpha_i)\frac{sA'(S_i)}{\chi_i} + 2s\left(\frac{1}{h_{i-1}\chi_i} + \frac{1}{h_i\chi_i}\right)\Psi'(S_i), \qquad i=1,\ldots,N-1, \tag{4.14}$$

$$\frac{\partial F_{i+1}}{\partial S_i} = (1-3\alpha_i)\frac{sA'(S_i)}{\chi_{i+1}} - 2\frac{s\Psi'(S_i)}{h_i\chi_{i+1}}, \qquad i=0,1,\ldots,N-1, \tag{4.15}$$

$$\frac{\partial F_N}{\partial S_N} = 1 + (1+3\alpha_{N-1})\frac{sA'(S_N)}{\chi_{N-1}} + 2\frac{s\Psi'(S_N)}{h_{N-1}\chi_{N-1}}, \tag{4.16}$$

$$\frac{\partial G_0}{\partial S_0} = 1 + (1-3\alpha_0)\frac{sA'(S_0)}{\chi_0} - 2\frac{s\Psi'(S_0)}{h_0\chi_0}, \tag{4.17}$$

$$\frac{\partial G_{i-1}}{\partial S_i} = (1+3\alpha_{i-1})\frac{sA'(S_i)}{\chi_{i-1}} + 2\frac{s\Psi'(S_i)}{h_{i-1}\chi_{i-1}}, \qquad i=1,2,\ldots,N, \tag{4.18}$$

$$\frac{\partial G_i}{\partial S_i} = 1 - (3\alpha_{i-1}+3\alpha_i)\frac{sA'(S_i)}{\chi_i} - 2s\left(\frac{1}{h_{i-1}\chi_i} + \frac{1}{h_i\chi_i}\right)\Psi'(S_i), \qquad i=1,\ldots,N-1, \tag{4.19}$$

$$\frac{\partial G_{i+1}}{\partial S_i} = -(1-3\alpha_i)\frac{sA'(S_i)}{\chi_{i+1}} + 2\frac{s\Psi'(S_i)}{h_i\chi_{i+1}}, \qquad i=0,1,\ldots,N-1, \tag{4.20}$$

$$\frac{\partial G_N}{\partial S_N} = 1 - (1+3\alpha_{N-1})\frac{sA'(S_N)}{\chi_{N-1}} - 2\frac{s\Psi'(S_N)}{h_{N-1}\chi_{N-1}}. \tag{4.21}$$

Equations (4.13) and (4.18) impose the following inequality for each parameter α_i in order to ensure that the Jacobian is nonnegative (explicit case) or is an M-matrix (implicit case).

$$\alpha_i \le \min\left\{ -\frac{2\Psi'(S_{i+1})}{3h_iA'(S_{i+1})} - \frac{1}{3}, 0\right\}, \qquad i=0,\ldots,N-1. \tag{4.22}$$

The inequalities for the parameters α_i may be interpreted as follows: Ideally, a set of parameters with minimum absolute value should be selected. If all the parameters could be chosen equal to zero, the discrete convection term in (4.9) would be $(A(S_{i+1}) - A(S_{i-1}))/2$. Consequently, this choice corresponds to central differencing. For points where the diffusion vanishes, a value of $-\frac{1}{3}$ must be selected. In this case, the discrete term would be $A(S_i) - A(S_{i-1})$, and one-point upstream weighting is in effect. By choosing the α_i's as small as possible in absolute value according to (4.22), one would generally do better than simple one-point upstream weighting. Therefore, each parameter α_i should be chosen to satisfy

$$\alpha_i = \begin{cases} \left|\dfrac{2\Psi'(S_{i+1})}{3h_iA'(S_{i+1})}\right| - \dfrac{1}{3} & \text{if } \left|\dfrac{2\Psi'(S_{i+1})}{3h_iA'(S_{i+1})}\right| - \dfrac{1}{3} < 0, \\ 0, & \text{otherwise}, \end{cases} \qquad i = 0,\dots,N-1. \tag{4.23}$$

The Jacobian G' for the implicit scheme has positive diagonal elements and negative off-diagonal elements provided the α_i's are chosen as in (4.23). By scaling with the diagonal matrix $D = \operatorname{diag}(\chi_i)$, the matrix DG' becomes columnwise diagonally dominant. The Jacobian is therefore an M-matrix for arbitrary time step s and mesh sizes h_i and we have unconditional monotonicity. If the diffusion vanishes one must do upstream weighting in order to ensure that the Jacobian is an M-matrix. When the fraction between diffusion and convection increases in absolute value, the scheme tends towards central differences (Galerkin). Note that given $\Psi'(S_i)$ and $A'(S_i)$ locally, (4.23) can be used to determine the maximum mesh size h_i for central differencing to be monotone.

We now discuss the explicit case. For this case, we get a restriction on the time step s. The Jacobian F' is nonnegative if $\partial F_i/\partial S_i \geq 0, i = 0, 1, \dots, N$. From the expressions (4.12), (4.14), and (4.16) it follows that

$$\begin{aligned} s \leq \min\Bigg\{ & \frac{h_0^2}{|2\Psi'(S_0)| + h_0|(1-3\alpha_0)A'(S_0)|}, \\ & \frac{h_{i-1}h_i\chi_i}{|\chi_i\Psi'(S_i)| + |h_{i-1}h_i(3\alpha_{i-1}+3\alpha_i)A'(S_i)|}, \\ & \frac{h_{N-1}^2}{|2\Psi'(S_N)| - h_{N-1}|(1+3\alpha_{N-1})A'(S_N)|} \Bigg\}, \end{aligned} \tag{4.24}$$

provided $|2\Psi'(S_N)| - h_{N-1}|(1+3\alpha_{N-1})A'(S_N)| > 0$. If this last expression is less than or equal to zero, the third bound in (4.24) disappears.

For linear parabolic problems on a uniform mesh the stability limit is $h^2/2\Psi'$. The monotonicity limit given by (4.24) indicates the same limit. For linear hyperbolic problems the stability limit, or CFL condition, is h/A'. If all parameters $\alpha_i = 0$, the same bound is forced by (4.24).

The restrictions on s and the α_i's given by (4.24) and (4.23) must hold for all solutions between two consecutive time levels. This is easy to verify for linear problems since the elements of the Jacobian are independent of the solution. For nonlinear problems, the two conditions must hold for all $S_i \geq \bar{S}_i$, $i = 0, 1, \dots, N$, where $\bar{S}_i$, $i = 0, 1, \dots, N$, is the solution at the previous time level (according to (2.4)).

The scheme developed so far is for Neumann-type boundary conditions where the flux is given at the boundaries. Dirichlet boundary conditions of the type $d(S) = e(t)$ are handled by writing either one or both of (4.8) and (4.10) in the form

$$\operatorname{sgn}(d'(S_i))d(S_i) = \operatorname{sgn}(d'(S_i))e(t), \qquad i = 0 \text{ and/or } i = N, \tag{4.25}$$

where $\operatorname{sgn}(x)$ is the sign of x. This gives positive contributions to the main diagonal of the Jacobian matrix. Diagonal dominance is ensured by the scaling of (4.25).

We will refer to the lumped Petrov–Galerkin scheme (4.8)–(4.10) as the LPG scheme. If the "upstream weighting" parameters α_i are chosen in the "optimal sense" given by (4.23), we will refer to that scheme as the OLPG scheme.

To guarantee monotonicity when $H(S) \neq 0$, one should use the Engquist–Osher flux [1].

The discretization of the second derivative in the traditional form (1.1) may cause nonmonotone schemes. This is because $D(S)$ may have local minima. Consider, for instance, the following discretization of the diffusion inner product in (4.7). The following diffusion term would then have appeared in (4.9) for a uniform mesh.

$$\begin{aligned} &\frac{1}{h}\left[-\int_0^1 D(S_h^{i-1})\frac{\partial S_h^{i-1}}{\partial \xi_{i-1}}\frac{\partial T_2^{i-1}}{\partial \xi_{i-1}}d\xi_{i-1} - \int_0^1 D(S_h^i)\frac{\partial S_h^i}{\partial \xi_i}\frac{\partial T_1^i}{\partial \xi_i}d\xi_i\right] \\ &= -\frac{1}{h}\left[(S_i - S_{i-1})\int_0^1 D(S_h^{i-1})\frac{\partial T_2^{i-1}}{\partial \xi_{i-1}}d\xi_{i-1} + (S_{i+1} - S_i)\int_0^1 D(S_h^i)\frac{\partial T_1^i}{\partial \xi_i}d\xi_i\right] \\ &\approx \frac{1}{h}\left[D(S_{i+\frac{1}{2}})(S_{i+1} - S_i) - D(S_{i-\frac{1}{2}})(S_i - S_{i-1})\right], \end{aligned} \tag{4.26}$$

$$S_{i-\frac{1}{2}} = \frac{1}{2}(S_{i-1} + S_i), \tag{4.27}$$

$$S_{i+\frac{1}{2}} = \frac{1}{2}(S_i + S_{i+1}). \tag{4.28}$$

Use of this discretization will make the elements of the Jacobian matrix dependent upon the local sign of $D'(S)$. This may destroy the monotonicity structure of the Jacobian matrices. This may be seen by considering the analogs of (4.14) and (4.19).

Using the PSI potential, the diffusive term in (4.9) is

$$\frac{1}{h}\left[(\Psi(S_{i+1}) - 2\Psi(S_i) + \Psi(S_{i-1}))\right]. \tag{4.29}$$

This discretization leads to a monotone scheme. We remark that (4.26) turns into the form of (4.29) for linear problems, for which $D = \text{const}$. To conclude, the discretization using the PSI potential constitutes a method for constructing monotone schemes for nonlinear parabolic operators.

5. Examples.

Example 1. The need for lumping. Consider the linear hyperbolic problem

$$\frac{\partial S}{\partial t} + \frac{\partial f}{\partial x} = 0 \tag{5.1}$$

$$f = S, \tag{5.2}$$

$$S(t)|_{x=0} = 1.0, \tag{5.3}$$

$$S(x)|_{t=0} = 0.2, \tag{5.4}$$

on a uniform mesh with all $\alpha_i = -4/11$.

Integrating the first innerproduct in (4.7) analytically gives the following analog scheme of (4.8)–(4.10):

$$S_0 = 1.0, \tag{5.5}$$

TABLE 1

Numerical solutions, example 1				
	$s = 1.0 \cdot 10^{-3}$		$s = 1.0 \cdot 10^{-2}$	
	$t = 2.0 \cdot 10^{-3}$		$t = 2.0 \cdot 10^{-2}$	
NODE	LPG	PG	LPG	PG
0	1.00	1.00	1.00	1.00
1	.251	.193	.547	.570
2	.202	.198	.323	.332
3	.200	.200	.240	.242
4	.200	.200	.212	.213
5	.200	.200	.204	.204
6	.200	.200	.201	.201

(5.6)
$$\frac{h}{s}\left(\frac{5}{66}(S^n_{i-1} - S^{n-1}_{i-1}) + \frac{2}{3}(S^n_i - S^{n-1}_i) + \frac{17}{66}(S^n_{i+1} - S^{n-1}_{i+1})\right)$$
$$-\frac{23}{22}S^n_{i-1} + \frac{12}{11}S^n_i - \frac{1}{22}S^n_{i+1}, = 0, \quad i = 1, 2, \ldots, N-1,$$

(5.7)
$$S_N = 0.20.$$

To ensure that the matrix $(J^{*\prime} + T)$ is of monotone type, we must impose the condition

(5.8)
$$\frac{5}{66}\frac{h}{s} - \frac{23}{22} \leq 0 \quad \Rightarrow \quad s \geq \frac{5}{69}h.$$

With $h = 1/33$, (5.8) yields $s \geq 5/2277 \approx 2.2 \cdot 10^{-3}$. Table 1 shows simulation results using the implicit lumped version (LPG) (4.8)–(4.10) and the implicit Petrov–Galerkin (PG) scheme (5.5)–(5.7) using different time steps. The LPG scheme is monotone, while the PG scheme tends to create some numerical oscillations when the lower bound for s forced by (5.8) is violated. For this case, such oscillations are not present as the "front" moves away from the boundary $x = 0$. The oscillations seem to be a consequence of the discontinuous initial condition forced by (5.3)–(5.4).

Example 2. Burgers equation. As an example of a nonlinear equation we consider the well-known Burgers equation. Here $A(S) = S^2/2$ and $D(S) = \epsilon = \text{const}$. Consider the initial condition $S = 0$ for $t = 0$. For a vanishing ϵ, the solution of the hyperbolic equation is two constant states separated by a shock moving from $x = 0$ to the right with speed $\frac{1}{2}$.

(5.9)
$$S(x,t) = \begin{cases} 1 & \text{if } x < \frac{1}{2}t, \\ 0 & \text{otherwise.} \end{cases}$$

The mesh spacing is constant, $h = 0.02$, $\epsilon = 0.0015$, and the OLPG scheme is chosen to minimize the need for artificial diffusion in accordance with (4.23). In Fig. 2 this solution is compared to the solutions from central differencing and from one-point upstream weighting. The reference solution using central differences (Galerkin) with h=0.002 is shown as a solid line. (For this mesh size the solution is monotone by (5.10).) The implicit version of the schemes was used, and the time step was controlled automatically with an overall maximum allowed solution change in each time step of 0.25. Since $A'(S) = S$ is strictly increasing for this case, the Jacobian is an M-matrix for all saturations $S_i \in [S_i^{n-1}, S_i^n]$ if the α_i's are chosen as

$$(5.10)\quad \alpha_i = \begin{cases} \left|\dfrac{2\epsilon}{3h_i\hat{S}_{i+1}}\right| - \dfrac{1}{3} & \text{if } \left|\dfrac{2\epsilon}{3h_i\hat{S}_{i+1}}\right| - \dfrac{1}{3} < 0, \\ 0, & \text{otherwise,} \end{cases} \qquad i = 0, 1, \ldots, N-1,$$

where $\hat{S}_{i+1} = \min[S_{i+1}^{n-1} + 0.25, 1.0]$. We note that the central-difference (Galerkin) scheme is not monotone, and one-point upstream weighting gives more smearing than the OLPG scheme.

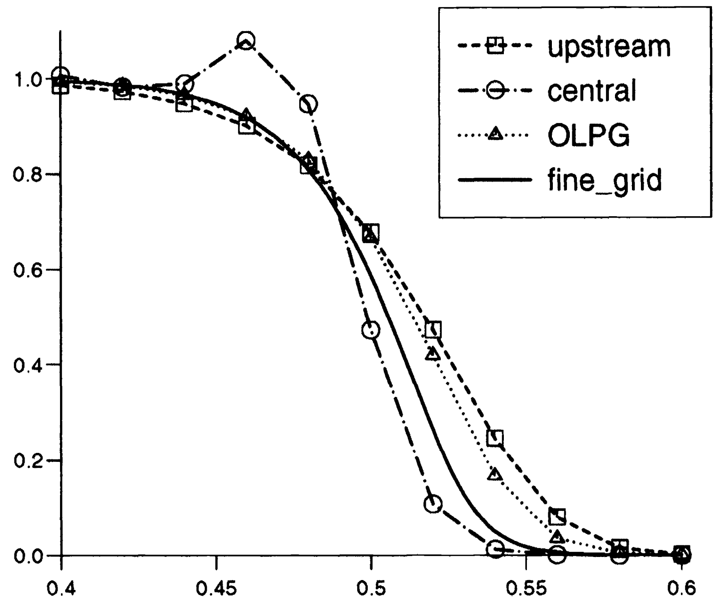


FIG. 2. *Burgers equation. Solution at* $t = 1.0$ *near* $x = 0.5$.

Example 3. The Buckley–Leverett equation with capillary diffusion. Here S represents the water saturation, and $0 \le S_{wc} \le S \le 1 - S_{or} \le 1$. For this problem $A(S)$ is an S-shaped function with $A(0) = 0$ and $A(1) = 1$. $D(S)$ is a bell-shaped function with $D(0) = 0$ and $D(1) = 1$. As initial and boundary conditions, we take

$$(5.11)\qquad S(x)_{t=0} = S_{wc}(x),$$

$$(5.12)\qquad f(S)_{x=0} = 1,$$

$$(5.13)\qquad f(S)_{x=1} = 0 \text{ as long as } S_{x=1} < 1 - S_{or},$$

or

$$S_{x=1} = 1 - S_{or}. \tag{5.14}$$

Such flux functions, together with the conditions (5.11)–(5.14), model immiscible incompressible displacement of oil by water in a porous core. Water is injected at a constant rate at the boundary $x = 0$. One or both of the fluids are produced at the boundary $x = 1$. More details about this phenomenon may be found in [20]. When the lumped Petrov–Galerkin method is used, a monotone solution is achieved both for Neumann and Dirichlet conditions. Figure 3 shows simulation results. As the advancing water front reaches the outlet $x = 1$, a boundary layer is formed by the capillary pressure effect; see [20].

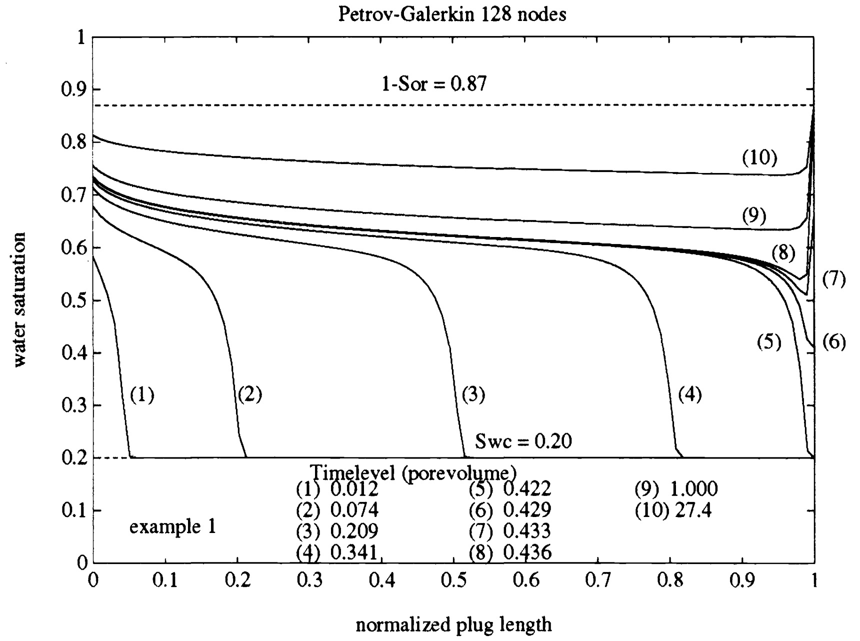


FIG. 3. *Simulation of a laboratory displacement experiment.*

6. Closing remarks. For the linear diffusion-convection problem $b\,\partial^2 S/\partial x^2 + a\,\partial S/\partial x = 0$ with Dirichlet boundary conditions, Hemker [13] used exponential fitting (to the true solution) of the differential operator in order to resolve the boundary layer for small b [13, p. 44]. Hemker used weighted finite differences and showed that the following choice of the parameters α_i gave exact solution at the nodes

$$\alpha_i = \left|\frac{2b}{3h_i a}\right| - \frac{1}{3}\coth\left(\frac{h_i a}{2b}\right). \tag{6.1}$$

This result was originally given by Il'in [17].

We remark that as the ratio between convection and diffusion increases, $\coth(h_i a/2b)$ approaches unity, and we are left with (4.23). Equation (4.23) may also be obtained by the approximate symmetrization process of Barrett and Morton [2]. Dahle, Espedal, and

Ewing [7] obtained the choice of parameters given by (6.1). In their work, the constants $a = A'$ and $b = \Psi'$ were representative values on each element (averages).

To conclude, we point out that the choice of parameters α_i given by (4.23) adds more artificial diffusion to the scheme than (6.1), but with (4.23) we ensure monotonicity for nonlinear problems with variable mesh spacing and for both Dirichlet and Neumann boundary conditions.

The test for monotonicity is not restricted to Petrov–Galerkin methods, but could be applied to any two-level in-time difference scheme. It could also be extended to higher dimensions. It is straightforward to show that the standard five-point scheme (two dimensions) and seven-point scheme (three dimensions) using upstream weighting result in monotone preserving schemes. The parabolic equation (1.1) is used in modelling of incompressible two-phase flow. In two and three dimensions, the equation is coupled to an elliptic pressure-velocity equation. For a fully implicit scheme the analysis may not be straightforward, since pressure appears as an additional variable. In higher dimensions, cross-derivative terms may also complicate the analysis.

Suitable choices of test functions for Petrov–Galerkin methods represent a problem. Most work has been done in two dimensions and for bilinear rectangular elements. Barrett and Morton discussed the exponential test functions of Hemker [13], the discontinuous test functions of Hughes and Brooks [15], and the quadratic test functions of Heinrich et al. [14]: The exponential test functions appear difficult to extend. The discontinuous test functions form the basis of a nonconforming method known as the streamline diffusion method. This method avoids crosswind diffusion. The test functions used by Heinrich et al. are products of the corresponding one-dimensional test functions and represent a simple extension. For triangular elements, Huyakorn [16] indicated suitable test functions. Recently, there has been great interest in the Eulerian Lagrangian localized adjoint method (ELLAM) [5]. The test functions are constructed (elementwise) from the local adjoint equation. Also, the weak form is considered in space and time so that the test functions become time-dependent. The method gives promising results, although some small numerical oscillations are reported. Its performance for variable coefficient and nonlinear problems remains to be seen.

To use the ideas presented in this paper, the test functions must be chosen in such a way that the Jacobian satisfies the required monotonicity properties. If this is feasible, one could minimize the amount of artificial diffusion. But the generalization of the ideas needs to be studied in more detail.

REFERENCES

[1] I. Aavatsmark, *Modellierung von Verdrängungsvorgängen in Laborversuchen*, Z. Angew. Math. Mech., 71 (1991), pp. 341–354.

[2] J. W. Barrett and K. W. Morton, *Approximate symmetrization and Petrov–Galerkin methods for diffusion-convection problems*, Comput. Methods Appl. Mech. Engrg., 45 (1984), pp. 97–122.

[3] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.

[4] Ø. Bøe, *Application of a Time Discretized Petrov–Galerkin Method for Simulation of Laboratory Displacement Experiments*, Bergen Scientific Centre Report Series 87/15, Bergen, Norway, 1987.

[5] M. Celia, T. F. Russell, I. Herrera, and R. E. Ewing, *An Eulerian–Lagrangian localized adjoint method for the advection-diffusion equation*, Adv. Water Resources, 13 (1990), pp. 187–205.

[6] L. Collatz, *Funktionalanalysis und numerische Mathematik*, Springer-Verlag, Berlin, 1964.

[7] H. K. Dahle, M. S. Espedal, and R. E. Ewing, *Characteristic Petrov–Galerkin sub-domain methods for convection-diffusion problems*, in Numerical Simulations in Oil Recovery, IMA Vol. 11, M. F. Wheeler, ed., Springer-Verlag, Berlin, 1988, pp. 77–88.

[8] L. DEMKOWICZ AND J. T. ODEN, *An adaptive Petrov–Galerkin finite element method for convection-dominated linear and nonlinear parabolic problems in two space variables*, Comput. Methods Appl. Mech. Engrg., 55 (1986), pp. 63–87.

[9] B. ENGQUIST AND S. OSHER, *Stable and entropy satisfying approximations for transonic flow calculations*, Math. Comput., 34 (1980), pp. 45–75.

[10] G. GILDING, *A nonlinear degenerate parabolic equation*, Ann. Scuola Norm. Sup. Pisa. Cl. Sci., 4 (1977), pp. 393–432.

[11] S. K. GODUNOV, *Raznostnyj metod čislennogo rasčeta razryvnyh rešenij uravnenij gidrodinamiki*, Mat. Sb., 47 (89) (1959), pp. 271-306.

[12] A. HARTEN, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), p. 357.

[13] P. W. HEMKER, *A Numerical Study of Stiff Two-Point Boundary Problems*, Mathematical Centre Tracts, No. 80, Mathematisch Centrum, Amsterdam, 1977.

[14] J. C. HEINRICH, P. S. HUYAKORN, A. R. MITCHELL, AND O. C. ZIENKIEWICZ, *An upwind finite element scheme for two-dimensional convective transport equations*, Internat. J. Numer. Methods Engrg., 11 (1977), pp. 131–143.

[15] T. J. R. HUGHES AND A. BROOKS, *A theoretical framework for Petrov–Galerkin methods with discontinuous weighting functions: Application to the streamline-upwind procedure*, in Finite Elements in Fluids, 4, R. H. Gallagher, D. H. Norrie, J. T. Oden, and O. C. Zienkiewicz, eds., John Wiley & Sons, New York, 1982.

[16] P. S. HUYAKORN, *Solution of Steady-State, Convective Transport Equation Using an Upwind Finite Element Scheme*, Appl. Math. Modelling, (1977), p. 187.

[17] A. M. ILIN, *Differencing scheme for a differential equation with a small parameter affecting the highest derivative*, Math. Notes Acad. Sci. USSR., 6 (1969), pp. 596–602.

[18] P. D. LAX, *Weak solutions of nonlinear hyperbolic equations and their numerical computation*, Comm. Pure Appl. Math., 7 (1954), pp. 159–193.

[19] A. R. MITCHELL AND D. T. GRIFFITHS, *The Finite Difference Method in Partial Differential Equations*, John Wiley & Sons, New York, 1980.

[20] C. M. MARLE, *Multiphase flow in porous media*, Éditions Technip, Paris, 1981.

[21] T. MEIS AND U. MARCOWITZ, *Numerische Behandlung partieller Differentialgleichungen*, Springer-Verlag, Berlin, 1978.

[22] O. A. OLEJNIK, A. S. KALAŠNIKOV, AND Y. ZHOU, *Zadača Cauchy i kraevye zadači dlja uravnenij tipa nestacionarnoj fil'tracii*, Izv. Akad. Nauk SSSR Ser. Mat., 22 (1958), pp. 667–704.

[23] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[24] R. SANDERS, *On convergence of monotone difference schemes with variable spatial differencing*, Math. Comput., 40 (1983), pp. 91–106.

[25] P. K. SWEBY, *High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws*, SIAM J. Numer. Anal., 21 (1984), pp. 995–1011.

[26] B. VAN LEER, *Towards the ultimate conservative scheme*: IV. *A new approach to numerical convection*, J. Comput. Phys., 23 (1977), pp. 276–299.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1072–1088, September 1993

MULTILEVEL ITERATION FOR MIXED FINITE ELEMENT SYSTEMS WITH PENALTY*

ZHIQIANG CAI[†], CHARLES I. GOLDSTEIN[‡], AND JOSEPH E. PASCIAK[‡]

Abstract. In this paper, the authors consider the solution of the discrete systems that arises when a mixed finite element approach is used to approximate the solution of second-order elliptic boundary value problems. By the introduction of a penalty parameter, these equations can be approximated by the solution of a symmetric and positive definite penalty system on the velocity subspace. Iterative procedures are developed and analyzed for this penalty system based on the hierarchical basis approach as well as on the standard multigrid approach. Finally, numerical experiments are presented that illustrate the convergence behavior suggested by the theory.

Key words. mixed approximation of elliptic boundary value problems, multigrid techniques, multilevel, preconditioners, hierarchical preconditioners

AMS subject classifications. primary 65N30; secondary 65F10

1. Introduction.

In this paper, we shall be concerned with solving the discrete system of equations that result from mixed approximation of second-order elliptic boundary value problems. Specifically, we consider mixed approximation based on the "so-called" Raviart–Thomas [23] approximation spaces. Such approximations lead to the solution of linear systems involving block matrices of the form

$$M = \begin{pmatrix} A & B^t \\ B & 0 \end{pmatrix}. \tag{1.1}$$

Here A is symmetric and positive definite and B^t denotes the transpose of the matrix B. The matrix M is clearly symmetric and indefinite.

There have been a number of approaches discussed in the literature for solving these discrete systems (e.g., [6], [13], [15], [16], [21], and references in [24]). Instead of solving this system directly, we consider solving the penalty approximation to it proposed in [2]. This approximation involves the use of a small parameter ϵ and results in a matrix problem involving the block matrix

$$\begin{pmatrix} A & B^t \\ B & -\epsilon I \end{pmatrix}. \tag{1.2}$$

Obvious manipulations show that problems involving (1.2) can be reduced to the solution of a matrix system involving the first block of unknowns and the matrix

$$A + \epsilon^{-1} B^t B. \tag{1.3}$$

This system, although symmetric and positive definite, can have a rather large condition number depending on ϵ. In this paper, we shall consider multilevel iterative techniques for solving linear systems with matrix (1.3).

*Received by the editors March, 18, 1991; accepted for publication (in revised form) September 6, 1992. 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 nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

[†]Department of Mathematics, University of Southern California, Los Angeles, California 90089-1113.

[‡]Department of Applied Science, Brookhaven National Laboratory, Upton, New York 11973.

Multilevel approaches have had a long and successful history applied to standard finite element and finite difference approximation of second-order boundary value problems (cf., for example, [1], [3], [7], [9], [10], [12], [17], [18], [20], [25], and the references in [22]). Two basic approaches have emerged. The standard multigrid approach involves the combination of simple iterative methods on a sequence of successively coarser approximation subspaces of the given approximation space (cf. [7], [9], [18], [20], and the references in [22]). In contrast, the hierarchical approach involves the decomposition of the original approximation subspace into "hierarchical components" associated with the successively coarser approximation subspaces. In the hierarchical approach, subspace iteration only involves the degrees of freedom associated with the hierarchical component [3], [25]. We shall consider both the multigrid and hierarchical approaches for solving (1.3) in this paper. In particular, we shall be interested in understanding how the parameter ϵ influences the rate of convergence for the respective methods.

In §3, we develop and analyze a hybrid hierarchical preconditioner for (1.3). This development involves two distinct phases. In the first phase, the original matrix (1.3) is preconditioned by the corresponding block diagonal matrix associated with the hierarchical subspace decomposition. The second phase involves replacing the hierarchical blocks in the block diagonal matrix by easily solved preconditioners. In contrast to the standard finite element and finite difference hierarchical matrices, the hierarchical blocks associated with (1.3) are not well conditioned. To develop efficient preconditioners for the hierarchical block problems, we use a variation of static condensation (or local domain decomposition). We then show that this "hybrid" hierarchical approach leads to a preconditioned system for (1.3) with condition number that is independent of ϵ and the mesh size of the fine grid approximation. It does, however, depend on the number of coarser grid approximation levels.

In §4, we develop and analyze the multigrid method applied to (1.3). The multigrid method is simple to apply since (1.3), in the case of Raviart–Thomas elements, fits into the standard variational multigrid framework [4], [7], [20]. We provide what appears to be a very crude analysis for the resulting procedure leading to estimates which deteriorate rather badly when ϵ or the fine grid mesh size become small. However, these estimates are probably sharp since they are in agreement with the results of numerical experiments reported in §5. We conclude that the natural multigrid approach is not very effective when directly applied to (1.3). We emphasize, however, that alternative multigrid methods have been proposed for other formulations of problem (1.1) (see [5], [11], [19], and references cited therein).

2. Penalty finite element method. In this section, we first define the continuous and discrete mixed variational problems. Next, the penalty formulation of this mixed method is given. Finally, we discuss the lowest-order Raviart–Thomas finite element space on triangles and the resulting system of linear equations.

Let Ω be a bounded open subset of R^2 with boundary Γ. Consider the boundary value problem

$$\begin{aligned} -\nabla\cdot(k\nabla p) &= f \quad \text{in} \quad \Omega, \\ p &= 0 \quad \text{on} \quad \Gamma, \end{aligned} \tag{2.1}$$

where $f \in L^2(\Omega)$ and $k(x) = k(x_1, x_2)$ is bounded from above and below by some positive constants.

We shall use the following space to define the mixed variational problem. Let

$$H(\text{div};\Omega) \equiv \{\mathbf{v} \in [L^2(\Omega)]^2 | \nabla\cdot\mathbf{v} \in L^2(\Omega)\}.$$

$H(\mathrm{div};\Omega)$ is a Hilbert space with norm given by

$$\|\mathbf{v}\|^2_{H(\mathrm{div};\Omega)} \equiv \|\mathbf{v}\|^2_{L^2(\Omega)} + \|\nabla\cdot\mathbf{v}\|^2_{L^2(\Omega)}. \tag{2.2}$$

We set $\mathbf{u} \equiv k\nabla p$ where p is the solution of (2.1). Note that the pair $(\mathbf{u}, p)$ satisfies

$$\begin{aligned} (k^{-1}\mathbf{u}, \mathbf{v}) + (p, \nabla\cdot\mathbf{v}) = 0 \quad \forall \quad \mathbf{v} \in H(\mathrm{div};\Omega),\\ (\nabla\cdot\mathbf{u}, q) = -(f, q) \quad \forall \quad q \in L^2(\Omega), \end{aligned} \tag{2.3}$$

where $(\cdot,\cdot)$ denotes the inner product in $L^2(\Omega)$ or $[L^2(\Omega)]^2$.

To discretize the mixed formulation (2.3), we assume that we are given two finite element subspaces

$$M \subset H(\mathrm{div};\Omega) \quad \text{and} \quad W \subset L^2(\Omega),$$

defined on a quasi-uniform mesh with elements of size $O(h)$. The mixed approximation of $(\mathbf{u}, p)$ is defined to be the pair, $(\mathbf{u}^h, p^h) \in M \times W$, satisfying

$$\begin{aligned} (k^{-1}\mathbf{u}^h, \mathbf{v}) + (p^h, \nabla\cdot\mathbf{v}) = 0 \quad \forall \quad \mathbf{v} \in M,\\ (\nabla\cdot u^h, q) = -(f, q) \quad \forall \quad q \in W. \end{aligned} \tag{2.4}$$

We refer to [23] for the definition of a class of approximation subspaces M, W. In this manuscript, we shall only consider the lowest-order Raviart–Thomas spaces defined on a triangulation of Ω. These spaces will be described in detail later in this section.

Problem (2.4) can be restated in terms of operators. To this end, we define $A : M \to M$, $B : M \to W$ and $B^* : W \to M$ by

$$(A\mathbf{v}, \mathbf{v}') \equiv (k^{-1}\mathbf{v}, \mathbf{v}') \quad \forall \quad \mathbf{v}' \in M, \tag{2.5a}$$

$$(B\mathbf{v}, q) \equiv (\nabla\cdot\mathbf{v}, q) \quad \forall \quad q \in W, \tag{2.5b}$$

$$(B^* q, \mathbf{v}) \equiv (q, \nabla\cdot\mathbf{v}) \quad \forall \quad \mathbf{v} \in M. \tag{2.5c}$$

With this notation, (2.4) can be rewritten

$$\begin{pmatrix} A & B^* \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^h \\ p^h \end{pmatrix} = \begin{pmatrix} 0 \\ -f^h \end{pmatrix}, \tag{2.6}$$

where f^h denotes the $L^2(\Omega)$ orthogonal projection of f into W. System (2.6) is indefinite and the number of degrees of freedom, N, is given by

$$N = N_M + N_W \equiv \dim(M) + \dim(W).$$

The solution $(\mathbf{u}^h, p^h)$ can be approximated by solving a perturbed reduced system using a penalty approximation. Let $\varepsilon > 0$ be a small parameter and perturb (2.6) to obtain

$$\begin{pmatrix} A & B^* \\ B & -\varepsilon I \end{pmatrix} \begin{pmatrix} \mathbf{u}^h_\varepsilon \\ p^h_\varepsilon \end{pmatrix} = \begin{pmatrix} 0 \\ -f^h \end{pmatrix}. \tag{2.7}$$

Eliminating p_ε^h above gives rise to the following equation for $\mathbf{u}_\varepsilon^h$:

$$A^\varepsilon \mathbf{u}_\varepsilon^h \equiv (A + \varepsilon^{-1} B^* B)\mathbf{u}_\varepsilon^h = -\varepsilon^{-1} B^* f^h. \tag{2.8}$$

The operator A^ε is obviously positive definite symmetric and involves N_M degrees of freedom. Once (2.8) has been solved for $\mathbf{u}_\varepsilon^h$, p_ε^h can be computed by

$$p_\varepsilon^h = \varepsilon^{-1}(B\mathbf{u}_\varepsilon^h + f^h). \tag{2.9}$$

The penalty method was analyzed in [2] for a class of mixed methods. In particular, it follows from these results that for the finite element spaces in [23],

$$\begin{aligned} \|\mathbf{u} - \mathbf{u}_\varepsilon^h\|_{H(\mathrm{div};\Omega)} + \|p - p_\varepsilon^h\|_{L^2(\Omega)} \le C \Bigg(& \inf_{\mathbf{v}\in M} \|\mathbf{u} - \mathbf{v}\|_{H(\mathrm{div};\Omega)} \\ & + \inf_{q\in W} \|p - q\|_{L^2(\Omega)} + \varepsilon \|p\|_{L^2(\Omega)} \Bigg), \end{aligned} \tag{2.10}$$

where the constant C *is independent of both h and ε.*

In the remainder of this section, we describe the lowest-order Raviart–Thomas spaces M and W and the resulting system of linear equations corresponding to (2.8), (2.9). To define M, first consider a reference triangle $\hat{T}$ in the $\hat{x} = (\hat{x}_1, \hat{x}_2)$-plane with vertices $(1,0)$, $(0,1)$, and $(0,0)$. Let $P_1(\hat{T})$ denote the set of polynomials on $\hat{T}$ of degree 1 or less. Define

$$V(\hat{T}) \equiv \{\hat{\mathbf{v}} \in P_1(\hat{T})^2 | \hat{\mathbf{v}} = (a + b\hat{x}_1, c + b\hat{x}_2)\},$$

where a, b, c are real numbers. $V(\hat{T})$ is a three-dimensional vector space whose elements are vector-valued functions with constant normal components on each edge of $\hat{T}$.

Now suppose that $\mathcal{T}$ is a partitioning of Ω into triangles T such that

$$\max_{T\in\mathcal{T}} \mathrm{diam}(T) \le h.$$

In the subsequent discussion, we shall always consider these triangles to be closed sets including their boundary. We assume that the intersection of any two distinct triangles either consists of a common side, a common vertex, or is empty. Any $T \in \mathcal{T}$ is the image of $\hat{T}$ under an affine transformation

$$x = F_T(\hat{x}) = B_T \hat{x} + b_T,$$

with B_T a two-by-two matrix. Define

$$V(T) \equiv \{\mathbf{v} = B_T(\hat{\mathbf{v}} \circ F_T^{-1}), \hat{\mathbf{v}} \in V(\hat{T})\}.$$

We now define

$$M \equiv \{\mathbf{v} \in H(\mathrm{div};\Omega) \; : \; \mathbf{v}|_T \in V(T) \quad \forall \quad T \in \mathcal{T}\}.$$

Note that the normal components of $\mathbf{v} \in M$ across the edges of triangles in $\mathcal{T}$ are continuous. The dimension N_M is equal to the number of edges in the triangulation of Ω. Choose a unit normal direction $\vec{n}_E$ on each edge E. In the following, the normal components across E of functions in M are taken with respect to $\vec{n}_E$. A convenient basis for M consists of the set of functions φ_E that are defined to have a normal component equal to one on E and a normal component of zero on the remaining edges.

Finally, the space W consists of piecewise constants with respect to the triangulation $\mathcal{T}$. Clearly,

$$\nabla \cdot \mathbf{v} \in W \quad \forall \quad \mathbf{v} \in M.$$

A simple basis for W consists of the set of functions $\{p_i\}$ such that $p_i = 1$ on the ith triangle and vanishes on all other triangles in $\mathcal{T}$. Since the support of p_i is contained in only one triangle, it follows that the mass matrix with respect to this basis is diagonal. Hence once (2.8) is solved for $\mathbf{u}_\varepsilon^h$, we may compute p_ε^h from (2.9) by inverting a diagonal matrix of degree N_W, where N_W is the number of triangles.

It is easy to see that the stiffness matrix corresponding to (2.8) is sparse, symmetric, and positive definite. Moreover, it can be shown that the condition number of this system is on the order of $\epsilon^{-1}h^{-2}$. In the remaining sections, we discuss various methods for solving this system.

3. Hierarchical preconditioner. In this section, we derive and analyze a hierarchical preconditioner for the penalty equation (2.8). This preconditioner is defined in terms of a hierarchical decomposition of the space $\mathcal{M}$. The first step is to analyze the block hierarchical form as a preconditioner for (2.8). The stiffness matrix associated with A^ε restricted to the hierarchical subspace, which forms the hierarchical block, is still nontrivial to invert. The final preconditioner involves replacing these hierarchical blocks by preconditioners. The main results provide estimates for the condition number of the resulting preconditioned operators for (1.3) (see Theorem 3.1 and Theorem 3.2).

We start with a coarse initial triangulation $\mathcal{T}_0$ of the domain Ω and construct a nested family,

$$\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_J \equiv \mathcal{T},$$

of triangulations of the domain Ω by subdividing any triangle of $\mathcal{T}_j$ into four congruent subtriangles to obtain $\mathcal{T}_{j+1}$. For each $j = 0, 1, \dots, J$, corresponding to the triangulation $\mathcal{T}_j$, we consider the lowest-order Raviart–Thomas velocity space $\mathcal{M}_j$ defined in §2. Let $\mathcal{E}_j$ denote the set of edges of all triangles in $\mathcal{T}_j$, and let $\mathcal{E} = \mathcal{E}_J$. It was noted in §2 that any vector function $\mathbf{v}$ in $\mathcal{M}_j$ is uniquely determined by its normal component, α_E, on each $E \in \mathcal{E}_j$. For any vector function $\mathbf{v} \in \mathcal{M}_j$,

$$\mathbf{v} = \sum_{E \in \mathcal{E}_j} \alpha_E \varphi_E, \tag{3.1}$$

where φ_E is the basis function in $\mathcal{M}_j$ associated with the edge $E \in \mathcal{E}_j$. It is easy to see that for any triangle $T \in \mathcal{T}_j$, there exist constants $c_1 > 0$ and $c_2 > 0$ independent of the mesh size such that

$$c_1 \int_T \mathbf{v} \cdot \mathbf{v}\, dx \le \sum_{\substack{E \in \mathcal{E}_j \\ E \subset T}} \alpha_E^2 \mid T \mid \le c_2 \int_T \mathbf{v} \cdot \mathbf{v}\, dx \quad \forall \quad \mathbf{v} \in \mathcal{M}_j. \tag{3.2}$$

Here and henceforth, we use c, with or without subscript, to denote generic positive constants independent of ε, J, and the mesh size. $|T|$ (respectively, $|E|$) denotes the area of T (respectively, the length of E). Because of the construction of the nested family of triangulations, it is easy to check that the family of spaces $\{\mathcal{M}_j\}$ is nested, i.e.,

$$\mathcal{M}_0 \subset \mathcal{M}_1 \subset \cdots \subset \mathcal{M}_J \equiv \mathcal{M}.$$

For any vector functions $\mathbf{u}, \mathbf{v} \in \mathcal{M}$, define the bilinear form

$$A^{\varepsilon}(\mathbf{u}, \mathbf{v}) = (a\mathbf{u}, \mathbf{v}) + \frac{1}{\varepsilon}(\nabla \cdot \mathbf{u}, \nabla \cdot \mathbf{v}), \tag{3.3}$$

where $a = k^{-1}$. We note that for the Raviart–Thomas elements,

$$(A^{\varepsilon}\mathbf{u}, \mathbf{v}) = A^{\varepsilon}(\mathbf{u}, \mathbf{v}) \quad \forall \quad \mathbf{u}, \mathbf{v} \in \mathcal{M}.$$

Our final aim is to derive a preconditioner for the penalty form A^{ε} on $\mathcal{M} \times \mathcal{M}$. First, we introduce the operators

$$I_j : \mathcal{M} \longrightarrow \mathcal{M}_j, \quad j = 0, 1, \dots, J$$

as follows: For any vector function $\mathbf{v} \in \mathcal{M}$, we write

$$\mathbf{v} = \sum_{e \in \mathcal{E}} \alpha_e \varphi_e, \tag{3.4}$$

where α_e and φ_e are the normal component of $\mathbf{v}$ and the basis function for $\mathcal{M}$ related to the edge $e \in \mathcal{E}$, respectively. Then $I_j\mathbf{v} \in \mathcal{M}_j$ is defined by

$$I_j\mathbf{v} = \sum_{E \in \mathcal{E}_j} \alpha_E \varphi_E, \tag{3.5}$$

where φ_E is the basis function for $\mathcal{M}_j$ associated with the edge $E \in \mathcal{E}_j$ and α_E is defined to be the mean value of the normal components of $\mathbf{v}$ on E, i.e.,

$$\alpha_E = \frac{1}{m_j} \sum_{\substack{e \in \mathcal{E} \\ e \subset E}} \alpha_e. \tag{3.6}$$

Here $m_j = 2^{J-j}$. Note that α_E is just the mean value of the coefficients $\{\alpha_e\}$ corresponding to edges $e \in \mathcal{E}$ with $e \subset E$. For convenience, let $I_{-1} = 0$. Clearly, $I_J = I$ where I is the identity operator on $\mathcal{M}$. With the above definitions, the hierarchical subspace $\tilde{\mathcal{M}}_j$ is defined to be the image of $I_j - I_{j-1}$, i.e.,

$$\tilde{\mathcal{M}}_j = \{\mathbf{w} \mid \mathbf{w} = (I_j - I_{j-1})\mathbf{v}, \mathbf{v} \in \mathcal{M}\}.$$

Remark 3.1. $\tilde{\mathcal{M}}_j$ is the set of functions in $\mathcal{M}_j$ whose normal components have zero mean value on the edges in $\mathcal{E}_{j-1}$.

For any vector function $\mathbf{v}$ in $\mathcal{M}$, we have the hierarchical decomposition,

$$\mathbf{v} = \sum_{j=0}^{J} \mathbf{v}_j, \quad \mathbf{v}_j = (I_j - I_{j-1})\mathbf{v}. \tag{3.7}$$

We will first consider preconditioning the original bilinear form $A^{\varepsilon}(\mathbf{u}, \mathbf{v})$ by the block hierarchical form

$$B^{\varepsilon}(\mathbf{u}, \mathbf{v}) = \sum_{j=0}^{J} A^{\varepsilon}(\mathbf{u}_j, \mathbf{v}_j). \tag{3.8}$$

The subsequent analysis shows that the condition number of the preconditioned system is bounded by $cJ2^J$, where c is a constant independent of ε, J, and the mesh size.

LEMMA 3.1. *For any vector function* $\mathbf{v} \in \mathcal{M}$, *we have*

$$\sum_{j=0}^{J} | \nabla \cdot I_j \mathbf{v} |^2_{L^2(\Omega)} \le (J+1) \, | \nabla \cdot \mathbf{v} |^2_{L^2(\Omega)} \,. \tag{3.9}$$

Proof. Fix $j \in [0,1,\dots,J]$ and let $\mathbf{v} \in \mathcal{M}$. Let $E \in \mathcal{E}_j$. It follows from the definition of I_j that

$$\int_E I_j \mathbf{v} \cdot \vec{n} \, ds = \sum_{\substack{e \in \mathcal{E} \\ e \subset E}} \int_e \mathbf{v} \cdot \vec{n} \, ds.$$

Since the divergence of $I_j\mathbf{v}$ on any triangle $T \in \mathcal{T}_j$ is constant, by using the Gauss divergence theorem and the above equality, we have for $x \in T$

$$\begin{aligned} \nabla \cdot I_j \mathbf{v}(x) &= \frac{1}{|T|} \int_{\partial T} I_j \mathbf{v} \cdot \vec{n} \, ds \\ &= \frac{1}{|T|} \sum_{\substack{E \in \mathcal{E}_j \\ E \subset T}} \sum_{\substack{e \in \mathcal{E} \\ e \subset E}} \int_e \mathbf{v} \cdot \vec{n} \, ds \\ &= \frac{1}{|T|} \int_T \nabla \cdot \mathbf{v} \, dx. \end{aligned}$$

Hence, we have

$$\int_T | \nabla \cdot I_j \mathbf{v} |^2 \, dx \le \frac{1}{|T|} \left(\int_T | \nabla \cdot \mathbf{v} | \, dx \right)^2 \le \int_T |\nabla \cdot \mathbf{v}|^2 \, dx.$$

The lemma immediately follows by summing the above inequality.

LEMMA 3.2. *For any vector function* $\mathbf{v} \in \mathcal{M}$, *we have*

$$\sum_{j=0}^{J} (a I_j \mathbf{v}, I_j \mathbf{v}) \le c 2^J (a\mathbf{v}, \mathbf{v}), \tag{3.10}$$

where c is a constant independent of J and the mesh size.

Proof. Fix $j \in \{0,1,\dots,J\}$, $\mathbf{v} \in \mathcal{M}$, and write $\mathbf{v}$ as in (3.4). $I_j\mathbf{v}$ is thus defined by (3.5). By (3.2), we have

$$\int_T a |I_j \mathbf{v}|^2 dx \le c \, | T | \sum_{\substack{E \in \mathcal{E}_j \\ E \subset T}} \alpha_E^2.$$

By the definition of α_E in (3.6) and the Cauchy-Schwarz inequality,

$$\begin{aligned} \int_T a |I_j \mathbf{v}|^2 \, dx &\le c \frac{|T|}{m_j} \sum_{\substack{E \in \mathcal{E}_j \\ E \subset T}} \sum_{\substack{e \in \mathcal{E} \\ e \subset E}} \alpha_e^2 \le c m_j \sum_{\substack{t \in \mathcal{T} \\ t \subset T}} \left(\sum_{\substack{e \in \mathcal{E} \\ e \subset t}} \alpha_e^2 \right) | t | \\ &\le c 2^{J-j} \int_T a |\mathbf{v}|^2 \, dx. \end{aligned}$$

Hence,

$$(aI_j\mathbf{v}, I_j\mathbf{v}) \le c2^{J-j}(a\mathbf{v}, \mathbf{v}).$$

The lemma follows by summing the above inequality from 0 to J.

From the definition of $B^\varepsilon(\cdot,\cdot)$, it immediately follows that

$$A^\varepsilon(\mathbf{v},\mathbf{v}) \le (J+1)B^\varepsilon(\mathbf{v},\mathbf{v}).$$

Moreover, Lemmas 3.1 and 3.2 imply that

$$B^\varepsilon(\mathbf{v},\mathbf{v}) \le c2^J A^\varepsilon(\mathbf{v},\mathbf{v}).$$

We have proved the following theorem.

THEOREM 3.1. *For any vector function* $\mathbf{v} \in \mathcal{M}$, *we have*

$$c2^{-J}B^\varepsilon(\mathbf{v},\mathbf{v}) \le A^\varepsilon(\mathbf{v},\mathbf{v}) \le (J+1)B^\varepsilon(\mathbf{v},\mathbf{v}), \tag{3.11}$$

where c is a constant independent of ε, J, *and the mesh size.*

The above theorem shows that the form B^ε can be used to effectively precondition the form A^ε on the subspace $\mathcal{M}$ as long as J is not too large. This, however, does not lead to an effective algorithm in practice since the block hierarchical problems are not easily solved. To define a computationally effective preconditioner, we replace these block hierarchical problems by preconditioners.

Preconditioning the hierarchical problems amounts to preconditioning the bilinear form $A^\varepsilon(\cdot,\cdot)$ on the hierarchical subspaces $\tilde{\mathcal{M}}_j$ $\quad (j = 1, 2, \dots, J)$. We solve the problem corresponding to $A^\varepsilon(\cdot,\cdot)$ or a preconditioner for it on $\tilde{\mathcal{M}}_0 = \mathcal{M}_0$.

For $j > 0$, the preconditioner for the bilinear form $A^\varepsilon(\cdot,\cdot)$ on $\tilde{\mathcal{M}}_j$ is developed by using a variation of domain decomposition (static condensation). We first define the subspace $\tilde{\mathcal{M}}_j^0$ of $\tilde{\mathcal{M}}_j$ to be the functions that have zero normal components on the edges in $\mathcal{E}_{j-1}$. That is,

$$\tilde{\mathcal{M}}_j^0 = \{\mathbf{v} \in \tilde{\mathcal{M}}_j \mid \mathbf{v}\cdot\vec{n} = 0 \quad \text{on} \quad E \quad \forall \quad E \in \mathcal{E}_{j-1}\}. \tag{3.12}$$

We next decompose vector functions in $\tilde{\mathcal{M}}_j$ as follows: For any $\mathbf{v} \in \tilde{\mathcal{M}}_j$, write

$$\mathbf{v} = \mathbf{v}_H + \mathbf{v}_P, \tag{3.13}$$

where $\mathbf{v}_P \in \tilde{\mathcal{M}}_j^0$ and satisfies

$$A^\varepsilon(\mathbf{v}_P, \mathbf{w}) = A^\varepsilon(\mathbf{v}, \mathbf{w}) \quad \forall \quad \mathbf{w} \in \tilde{\mathcal{M}}_j^0. \tag{3.14}$$

Thus, $\mathbf{v}_H$ satisfies the homogeneous equation

$$A^\varepsilon(\mathbf{v}_H, \mathbf{w}) = 0 \quad \forall \quad \mathbf{w} \in \tilde{\mathcal{M}}_j^0. \tag{3.15}$$

We clearly have that

$$A^\varepsilon(\mathbf{v}, \mathbf{w}) = A^\varepsilon(\mathbf{v}_H, \mathbf{w}_H) + A^\varepsilon(\mathbf{v}_P, \mathbf{w}_P) \tag{3.16}$$

holds for any vector functions $\mathbf{v}, \mathbf{w} \in \tilde{\mathcal{M}}_j$, where $\mathbf{w} = \mathbf{w}_H + \mathbf{w}_P$ is decomposed as in (3.13).

For $E \in \mathcal{E}_{j-1}$, define the basis function $\theta_E \in \tilde{\mathcal{M}}_j$ as follows:

θ_E has normal component plus and minus one on the respective two halves of E.

θ_E has zero normal component on the remaining edges in $\mathcal{E}_j$.

The subspace $\tilde{\mathcal{M}}'_j$ is defined as the space spanned by these functions $\{\theta_E\}$. Clearly, $\tilde{\mathcal{M}}'_j \oplus \tilde{\mathcal{M}}^0_j$ gives a direct sum decomposition of $\tilde{\mathcal{M}}_j$. Let $\mathbf{v}, \mathbf{w}$ in $\tilde{\mathcal{M}}_j$ be decomposed as in (3.13) and write

$$\mathbf{v}_H = \mathbf{v}'_H + \mathbf{v}^0_H, \qquad \mathbf{w}_H = \mathbf{w}'_H + \mathbf{w}^0_H, \tag{3.17}$$

where $\mathbf{v}'_H, \mathbf{w}'_H \in \tilde{\mathcal{M}}'_j$. Let $\{\alpha_E\}$ and $\{\beta_E\}$ denote the coefficients of $\mathbf{v}'_H$ and $\mathbf{w}'_H$ respectively, with respect to the basis $\{\theta_E\}$. The preconditioner for the hierarchical block is defined by replacing the $A^\varepsilon(\mathbf{v}_H, \mathbf{w}_H)$ term in (3.16) by

$$K(\mathbf{v}_H, \mathbf{w}_H) \equiv \sum_{E\in\mathcal{E}_{j-1}} \alpha_E \beta_E (a\theta_E, \theta_E). \tag{3.18}$$

Thus, we define the preconditioning form $\tilde{B}^\varepsilon_j(\cdot,\cdot)$ by

$$\tilde{B}^\varepsilon_j(\mathbf{v}, \mathbf{w}) = K(\mathbf{v}_H, \mathbf{w}_H) + A^\varepsilon(\mathbf{v}_P, \mathbf{w}_P). \tag{3.19}$$

We now prove the following lemma.

LEMMA 3.3. *For any vector function* $\mathbf{v} \in \tilde{\mathcal{M}}_j$, *there exist constants* $c_1 > 0$ *and* $c_2 > 0$ *independent of* ε, j, *and the mesh size such that*

$$c_1 \tilde{B}^\varepsilon_j(\mathbf{v}, \mathbf{v}) \le A^\varepsilon(\mathbf{v}, \mathbf{v}) \le c_2 \tilde{B}^\varepsilon_j(\mathbf{v}, \mathbf{v}). \tag{3.20}$$

Proof. By mapping the triangles of the jth grid onto a reference triangle, it is easily seen that

$$c'_3 K(\mathbf{v}_H, \mathbf{v}_H) \le h_j^2 \sum_{E\in\mathcal{E}_{j-1}} \alpha_E^2 \le c'_4 K(\mathbf{v}_H, \mathbf{v}_H). \tag{3.21}$$

Here α_E is the nodal coefficient in the decomposition of $\mathbf{v}'_H$ as discussed above and $h_j = 2^{-j}$. Consequently, the first inequality in (3.20) follows from (3.16), (3.18), (3.19), and (3.2).

To prove the second inequality, we note that for any $\chi \in \tilde{\mathcal{M}}^0_j$, we have

$$A^\varepsilon(\mathbf{v}_H, \mathbf{v}_H) \le A^\varepsilon(\mathbf{v}_H + \chi, \mathbf{v}_H + \chi).$$

Let T be any triangle in $\mathcal{T}_{j-1}$ and $t_i \subset T$, $i = 1,2,3,4$, be the triangles in $\mathcal{T}_j$ (see Fig. 3.1). By choosing χ_0 in $\tilde{\mathcal{M}}^0_j$ such that its normal component (pointing outward from t_i) on the edge $t_i \cap t_4 \in \mathcal{E}_j$, $i = 1,2,3$, is equal to

$$\alpha_i = -\frac{1}{|\, t_i \cap t_4 \,|} \int_{\partial t_i} \mathbf{v}_H \cdot \vec{n}_{t_i}\, ds,$$

we have

$$\int_{\partial t_i} \chi_0 \cdot \vec{n}_{t_i}\, ds = \int_{t_i\cap t_4} \chi_0 \cdot \vec{n}_{t_i}\, ds = -\int_{\partial t_i} \mathbf{v}_H \cdot \vec{n}_{t_i}\, ds. \tag{3.22}$$

Here and below, the notation $\vec{n}_t$ is used to denote the outward pointing normal with respect to the set t. Clearly, $\mathbf{v}_H + \chi_0$ is divergence free on t_i for $i = 1,2,3$. The divergence of $\mathbf{v}_H + \chi_0$ on t_4 is equal to

$$\begin{aligned}\nabla \cdot (\mathbf{v}_H + \chi_0)|_{t_4} &= |t_4|^{-1} \int_{\partial t_4} (\mathbf{v}_H + \chi_0) \cdot \vec{n}_{t_4}\, ds \\ &= |t_4|^{-1} \sum_{i=1}^{3} \int_{\partial t_i \setminus (t_i \cap t_4)} \mathbf{v}_H \cdot \vec{n}_{t_i}\, ds \\ &= |t_4|^{-1} \int_{\partial T} \mathbf{v}_H \cdot \vec{n}_T\, ds = 0.\end{aligned}$$

The last equality follows from Remark 3.1. Hence, we have

$$A^\epsilon(\mathbf{v}_H, \mathbf{v}_H) \le (a(\mathbf{v}_H + \chi_0), \mathbf{v}_H + \chi_0). \tag{3.23}$$

It is elementary to see, using (3.22), that the normal component of $\mathbf{v}_H + \chi_0$ on any edge in $\mathcal{E}_j$ can be written as a linear combination of the normal components of $\mathbf{v}_H$ on the neighboring edges in $\mathcal{E}_j$, which lie on the coarse grid edges in $\mathcal{E}_{j-1}$. The corresponding coefficients only depend on the angles in the coarsest grid triangulation. Thus, (3.2), (3.21), and (3.23) imply that

$$A^\epsilon(\mathbf{v}_H, \mathbf{v}_H) \le cK(\mathbf{v}_H, \mathbf{v}_H).$$

Now, the second inequality in (3.20) follows from (3.16) and (3.19). This completes the proof of the lemma.

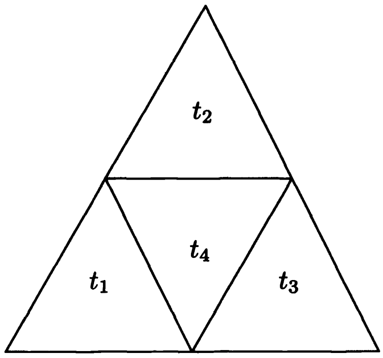


FIG. 3.1. *A typical triangle* $\mathcal{T}_{j-1}$.

To avoid solving coarse grid problems, we introduce a uniformly spectrally equivalent form $\tilde{B}_0(\cdot,\cdot)$ for $A^\epsilon(\cdot,\cdot)$. This means we assume that there are positive constants C_1 and C_2 satisfying

$$C_1 A^\epsilon(\mathbf{v},\mathbf{v}) \le \tilde{B}_0^\epsilon(\mathbf{v},\mathbf{v}) \le C_2 A^\epsilon(\mathbf{v},\mathbf{v}) \quad \forall \quad \mathbf{v} \in \mathcal{M}_0.$$

Let the quadratic form $\tilde{B}^\epsilon(\cdot,\cdot)$ be defined by

$$\tilde{B}^\epsilon(\mathbf{v},\mathbf{w}) = \sum_{j=0}^{J} \tilde{B}_j^\epsilon(\mathbf{v}_j, \mathbf{w}_j) \tag{3.24}$$

for any vector functions $\mathbf{v}, \mathbf{w} \in \mathcal{M}$ with $\mathbf{v}_j, \mathbf{w}_j$ as in (3.7). Using Theorem 3.1 and Lemma 3.3 we have the following theorem.

THEOREM 3.2. *For any vector function* $\mathbf{v} \in \mathcal{M}$, *there exist constants* $c_1 > 0$ *and* $c_2 > 0$ *independent of* ε, J, *and the mesh size such that*

$$c_1 2^{-J} \tilde{B}^{\varepsilon}(\mathbf{v},\mathbf{v}) \le A^{\varepsilon}(\mathbf{v},\mathbf{v}) \le c_2 J \tilde{B}^{\varepsilon}(\mathbf{v},\mathbf{v}). \tag{3.25}$$

Remark 3.2. The coarse grid bilinear form $\tilde{B}_0^{\varepsilon}(\ ,\ )$ may be defined, for example, by replacing the mass matrix corresponding to the first term on the right side of (3.3) by a suitable diagonal matrix D (cf. [13]). Since the mesh is quasi-uniform, we can choose D to be a weighted identity matrix. Other choices of D, such as the "lumped mass matrix" will possibly yield a better preconditioner.

Fix $j \in [1, \dots, J]$. We now demonstrate how to solve the problem: Given a linear functional F, find $\mathbf{v} \in \tilde{\mathcal{M}}_j$ such that

$$\tilde{B}_j^{\varepsilon}(\mathbf{v},\mathbf{w}) = F(\mathbf{w}) \quad \forall \quad \mathbf{w} \in \tilde{\mathcal{M}}_j. \tag{3.26}$$

If $\mathbf{w}$ is in $\tilde{\mathcal{M}}_j^0$, then $\mathbf{w}_H = 0$ and hence $\mathbf{v}_P$ can be obtained by solving

$$A^{\varepsilon}(\mathbf{v}_P,\mathbf{w}) = F(\mathbf{w}) \quad \forall \quad \mathbf{w} \in \tilde{\mathcal{M}}_j^0. \tag{3.27}$$

Note that the stiffness matrix corresponding to (3.27) consists of a block diagonal matrix where the blocks are 3×3 matrices connecting the three unknowns corresponding to normal components along the edges of the center triangle of $\mathcal{T}_j$ inside a triangle of $\mathcal{T}_{j-1}$.

With $\mathbf{v}_P$ now known, we are left to compute $\mathbf{v}_H$. Note that

$$\begin{aligned} K(\mathbf{v}_H,\mathbf{w}_H) &= F(\mathbf{w}) - A^{\varepsilon}(\mathbf{v}_P,\mathbf{w}_P) \\ &= F(\mathbf{w}) - A^{\varepsilon}(\mathbf{v}_P,\mathbf{w}) \end{aligned} \tag{3.28}$$

for any $\mathbf{w} \in \tilde{\mathcal{M}}'_j$. Since by (3.18), the left side of (3.28) gives a symmetric positive definite quadratic form on $\tilde{\mathcal{M}}'_j \times \tilde{\mathcal{M}}'_j$, (3.28) provides an equation for determining the normal components of $\mathbf{v}_H$ on the edges of the triangulation $\mathcal{T}_{j-1}$. The computation of these edge values involves the inversion of a diagonal matrix. Let $\mathbf{v}_B$ denote any extension of these edge values (e.g., the function that has vanishing normal components on the nodes of the jth grid that are in the interior of triangles $t_{j-1} \in \mathcal{T}_{j-1}$). Then, the difference $\mathbf{v}_I = \mathbf{v}_H - \mathbf{v}_B \in \tilde{\mathcal{M}}_j^0$ can be computed from

$$\begin{aligned} A^{\varepsilon}(\mathbf{v}_I,\mathbf{w}) &= A^{\varepsilon}(\mathbf{v}_H,\mathbf{w}) - A^{\varepsilon}(\mathbf{v}_B,\mathbf{w}) \\ &= -A^{\varepsilon}(\mathbf{v}_B,\mathbf{w}) \quad \forall \quad \mathbf{w} \in \tilde{\mathcal{M}}_j^0. \end{aligned} \tag{3.29}$$

We summarize the process for solving the problem (3.26) in the following algorithm.

ALGORITHM 3.1.

(1) Find $\mathbf{v}_P$ by solving (3.27).
(2) Compute the normal components of $\mathbf{v}_H$ on the coarse grid edges in $\mathcal{E}_{j-1}$ by (3.28).
(3) Find the normal components of $\mathbf{v}_H$ on the edges in $\mathcal{E}_j$, which do not lie on the edges in $\mathcal{E}_{j-1}$, by solving the problem (3.29).
(4) Compute $\mathbf{v} = \mathbf{v}_P + \mathbf{v}_H$.

Remark 3.3. The preceding results can be extended to grids that are not quasi-uniform, such as locally refined grids. Stability for the Raviart–Thomas spaces in such applications was considered in [14]. As an example, suppose we have a family of nested subdomains,

$$\Omega_J \subseteq \Omega_{J-1} \subseteq \ldots \subseteq \Omega_0 = \Omega.$$

To define the triangulation on each Ω_j, first assume we have a quasi-uniform coarse grid triangulation $\mathcal{T}_0$ of Ω. Next, given that $\mathcal{T}_{j-1}$ has been defined for $j\epsilon[1,\ldots,J]$, we define $\mathcal{T}_j$ by subdividing each triangle of $\mathcal{T}_{j-1}$ contained in Ω_j into four congruent subtriangles. We assume that $\partial\Omega_j$ aligns with the mesh corresponding to $\mathcal{T}_{j-1}$. The space $\mathcal{M}$ is now defined to be the Raviart–Thomas space defined as in §§2 and 3 corresponding to the triangles in $\mathcal{T}_J$. Suppose the triangle $T_{j-1} \subset \Omega_{j-1}$ intersects Ω_j on an edge E_{j-1} of T_{j-1}. Then there are two triangles, t_j^ℓ in Ω_j with edges e_j^ℓ on $\Omega_j \cap \Omega_{j-1}, \ell = 1, 2$, such that $e_j^1 \cup e_j^2 = E_{j-1}$. We define the nodal values for the "slave nodes" corresponding to e_j^1 and e_j^2 to be the nodal value corresponding to E_{j-1}. Hence $\mathbf{v}\cdot n$ is constant on $e_j^1 \cup e_j^2$ for each $\mathbf{v}\epsilon\mathcal{M}$. It is easily seen that this sequence of spaces is nested and Theorems 3.1 and 3.2 hold. The proofs are essentially the same as before.

4. A multigrid method. In this section, we first define a natural class of multigrid methods for solving the penalty equation (2.8) in the variational framework of §§2 and 3. We then prove some crude estimates that, when combined with results in [7] and [8], give rise to convergence bounds for the multigrid algorithms. These bounds only guarantee a very slow rate of convergence which deteriorates drastically as ϵ and h become small. The rates are in agreement with those observed in the numerical experiments of §5. Thus, the penalty system (2.8) provides an example where the natural multigrid formulation leads to an extremely ineffective algorithm.

This discussion is included to illustrate an example of a problem on which the most natural multigrid algorithm fails. Alternative multigrid algorithms not involving the penalty reformulation have been analyzed and have been shown to be more effective (see, e.g., [5], [11], and [19]).

Consider the nested family of spaces, $\{\mathcal{M}_j\}$, of vector functions defined in §3. Let A_j denote the positive definite symmetric operator on $\mathcal{M}_j$ defined by

$$(A_j\mathbf{v},\mathbf{w}) = A^\epsilon(\mathbf{v},\mathbf{w}) \quad \forall \quad \mathbf{v},\mathbf{w} \in \mathcal{M}_j, \tag{4.1}$$

where A^ϵ is defined by (3.3); $(\cdot,\cdot)$ again denotes the inner product on $L^2(\Omega)\times L^2(\Omega)$ with corresponding norm, $\|\cdot\|$. Let λ_j denote the largest eigenvalue of A_j. Define smoothing operators $\{R_j\}$ on $\mathcal{M}_j$ for $j = 1,\ldots,J$ by

$$R_j\mathbf{u} = \frac{1}{3}\sum_{E\in\mathcal{E}_j} d_E^{-1}(\mathbf{u},\varphi_E)\,\varphi_E \quad \forall \quad \mathbf{u} \in \mathcal{M}_j, \tag{4.2}$$

where $d_E = A^\epsilon(\varphi_E,\varphi_E)$ and φ_E is the basis element for M_j associated with the edge E defined in §3. R_j is clearly symmetric and positive definite on M_j equipped with the $[L^2(\Omega)]^2$ inner product. Furthermore, applying Theorem 3.1 [8] and Remark 3.2 of [8] shows that R_j satisfies the inequalities

$$(R_jA_j\mathbf{u},A_j\mathbf{u}) \le (A_j\mathbf{u},\mathbf{u}) \quad \forall \quad \mathbf{u} \in M_j, \tag{4.3}$$

and

$$\frac{\|\mathbf{u}\|^2}{\lambda_j} \le C_R(R_j\mathbf{u},\mathbf{u}) \quad \forall \quad \mathbf{u} \in \mathcal{M}_j. \tag{4.4}$$

Here λ_j denotes the largest eigenvalue of A_j and the constant C_R does not depend on ϵ, j or h. Estimates of the form (4.3) and (4.4) are standard hypotheses for smoothers used in analyses of multigrid procedures.

We define a symmetric multigrid operator $B_J : \mathcal{M} \longrightarrow \mathcal{M}$ by induction. Let P^0_{j-1} denote the L^2 projection operator from $\mathcal{M}_j$ onto $\mathcal{M}_{j-1}$.

ALGORITHM 4.1.

Suppose n_j is a given positive integer for each $j = 1, \ldots, J$. Set $B_0 = A_0^{-1}$. For $j > 0$, assume that B_{j-1} has been defined and define $B_j\mathbf{g}$ for $\mathbf{g} \in \mathcal{M}_j$ as follows:

(i) Set $\mathbf{u}^0 = 0$ and define $\mathbf{u}^l$ for $l = 1, \ldots, n_j$ by

$$\mathbf{u}_j^l = \mathbf{u}_j^{l-1} + R_j(\mathbf{g} - A_j\mathbf{u}_j^{l-1}). \tag{4.5}$$

(ii) Define $\mathbf{u}_j^{n_j+1} = \mathbf{u}_j^{n_j} + \mathbf{q}$, where $\mathbf{q} \in \mathcal{M}_{j-1} \subset \mathcal{M}_j$ is defined by

$$\mathbf{q} = B_{j-1}P^0_{j-1}(\mathbf{g} - A_j\mathbf{u}^{n_j}) \tag{4.6}$$

(iii) Set $B_j\mathbf{g} = \mathbf{u}_j^{2n_j+1}$, where $\mathbf{u}_j^l$ is defined for $l = n_j + 2, \ldots, 2n_j + 1$ by (4.5).

Remark 4.1. Because of the operator P^0_{j-1} appearing in (4.6), it appears that it is necessary to solve a mass matrix problem associated with the subspace $\mathcal{M}_{j-1}$ each time q is calculated. In fact, it is possible to implement the above algorithm in a way which avoids such mass matrix problems. A discussion of how this can be done is given in the appendix of [9].

Algorithm 4.1 defines a symmetric operator acting on $\mathcal{M}_j$. This is referred to as a symmetric V-cycle. If step (iii) is omitted and we set $B_j\mathbf{g} = \mathbf{u}_j^{n_j+1}$, we have a nonsymmetric V-cycle. For the standard V-cycle, n_j is the same for all j. If the coarse grid correction in step (ii) is performed twice, we have a W-cycle algorithm.

Remark 4.2. Often, multigrid algorithms are developed as iterative processes in contrast to the operator approach taken above. The convergence rate of the multigrid process is determined by the norm of the iterative reduction operator. In terms of the operators defined above, the iterative reduction operator is equal to $I - B_JA_J$. It can be shown in the case of (4.3) that $I - B_JA_J$ is a nonnegative operator in the $A^\epsilon(\cdot,\cdot)$ inner product for symmetric cycling algorithms. Thus, convergence estimates for the multigrid processes follow directly from estimates for the lowest eigenvalue of the operator B_JA_J.

There are a number of equivalent conditions used in the literature that form a basis for the analysis of multigrid methods. For our purposes, we use a so-called "regularity and approximation" estimate (cf. [4] and [7]). To describe this condition, let P_{j-1} denote the A^ϵ-projection from $\mathcal{M}_j$ into $\mathcal{M}_{j-1}$, i.e., $P_{j-1}\mathbf{v} = \mathbf{w}$, where $\mathbf{w}$ is the unique function in $\mathcal{M}_{j-1}$ satisfying

$$A^\epsilon(\mathbf{w}, \theta) = A^\epsilon(\mathbf{v}, \theta) \quad \forall \quad \theta \in \mathcal{M}_{j-1}.$$

The regularity and approximation assumption involves inequalities of the form

$$A^\epsilon((I - P_{j-1})\mathbf{u}, \mathbf{u}) \le c(\epsilon, h)\lambda_j^{-1} \|A_j\mathbf{u}\|^2_{L^2(\Omega)} \quad \forall \quad \mathbf{u} \in \mathcal{M}_j, \qquad j = 1, \ldots, J. \tag{4.7}$$

Under (4.3), (4.4) and (4.7), it is possible to show (e.g., [7]) that the norm of the multigrid reduction operator $I - B_JA_J$ is bounded by $(1 + C/c(\epsilon, h))^{-1}$ where the constant C does not depend on ϵ and h. This is equivalent to saying that the condition number of the system B_JA_J is bounded by $1 + c(\epsilon, h)/C$.

In the case of standard finite element and finite difference multigrid applications, it is often possible to prove that the inequalities corresponding to (4.7) hold with a constant $c(\epsilon, h)$ which can be bounded independently of h [1], [7]. Thus, in the standard applications, it is often possible to prove uniform rates of convergence for the multigrid process. Equivalently, it is possible to show that the condition number of the system $B_J A_J$ remains bounded independently of the mesh parameters. Moreover, in cases when (4.7) does not hold uniformly due to deficient elliptic regularity, it is still possible to show that the rate of convergence can deteriorate at worst like a power of the number of grid levels [7], [12].

Returning to the application at hand, we note that $I - P_{j-1}$ is bounded with norm one as an operator with respect to the norm $(A^\epsilon(\cdot,\cdot))^{1/2}$. Consequently,

$$A^\epsilon((I - P_{j-1})\mathbf{u}, \mathbf{u}) \le A^\epsilon(\mathbf{u}, \mathbf{u}) \le c_j^{-1} \|A_j \mathbf{u}\|^2_{L^2(\Omega)} \quad \forall \quad \mathbf{u} \in \mathcal{M}_j.$$

Here c_j denotes the smallest eigenvalue of A_j. Consequently, (4.7) holds with $c(\epsilon, h) = \lambda_j / c_j$. It is not difficult to see that c_j is of unit size, while λ_j grows proportional to $\epsilon^{-1} h_j^{-2}$ with $h_j = 2^{J-j} h$. The multigrid analysis then shows that the reduction rate for the multigrid process (see Remark 4.2) is bounded by

$$\frac{1}{1 + Ch^2\epsilon},$$

and that the condition number of $B_J A_J$ is bounded by

$$1 + (Ch^2\epsilon)^{-1}. \tag{4.8}$$

The numerical results presented in §5 suggest that this extremely pessimistic bound for multigrid is qualitatively sharp. Thus, estimates with lesser asymptotic order of growth are not likely to hold.

The above estimate is indeed pessimistic. Note that the condition number of the original system is λ_J / c_J. Thus, application of the multigrid method does not appear to change the asymptotic behavior of the number of iterations required for convergence. However, there may be a possible improvement associated with application of the multigrid preconditioner since some initial convergence acceleration was observed in unreported preconditioned conjugate gradient examples. It is quite likely that this improvement may be attributed to the preconditioner tending to cluster the eigenvalues without significantly improving the condition number.

5. Numerical results.

We present the results of numerical experiments illustrating the condition numbers of the preconditioned systems discussed earlier. The numerical results for the hierarchical method show that this approach is relatively effective. In contrast, the numbers reported suggest that the multigrid iteration gives rise to a system with an extremely large condition number. This is in agreement with the estimates provided in §4.

We consider the model problem (2.1) with coefficient $k(x) \equiv 1$ where Ω is the unit square $(0,1) \times (0,1)$. The problem is discretized by the mixed finite element method with penalty. Specifically, the domain Ω is first partitioned into $n \times n$ square subdomains of side length $1/n$. The $n \times n$ subsquares are then divided into pairs of triangles by connecting the bottom left and upper right corners. Subsequently, finer grids are developed as in §3, i.e., by dividing each triangle into four triangles formed by the edges of the original triangle and the lines connecting the centers of these edges. The approximation spaces $\{\mathcal{M}_j\}$ are defined to be the set of vector functions that are piecewise linear with respect to the triangulations with continuous normal components across the edges of the triangles as described in §3.

We will illustrate the rate of convergence of the hierarchical and multigrid preconditioned algorithms for solving the penalty equation (2.8), where $A^\epsilon(\cdot,\cdot)$ is defined in (3.3) with $a \equiv 1$. In the case of the hierarchical preconditioner, we numerically compute the largest and smallest generalized eigenvalues of the system

$$\lambda \tilde{B}^\epsilon(\mathbf{v},\cdot) = A^\epsilon(\mathbf{v},\cdot). \tag{5.1}$$

Theorem 3.2 provides the bounds $\lambda_{\max} \le c_2 J$ and $\lambda_{\min} \ge c_1 2^{-J}$ where c_1 and c_2 are the constants appearing in (3.25). The rate of convergence for preconditioned iterative schemes for solving (2.8) with preconditioner $\tilde{B}^\epsilon$ can be bounded in terms of the condition number $K(\tilde{B}^\epsilon A^\epsilon) = \lambda_{\max}/\lambda_{\min}$. In the case of preconditioned conjugate gradient iteration, the asymptotic reduction per step can be bounded by

$$\rho = \frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}}. \tag{5.2}$$

We note that both the hierarchical and multigrid methods require solving or preconditioning the coarse grid problem. In the reported runs, we always solved the coarse grid problems exactly.

Table 5.1 gives the largest and smallest eigenvalues of (5.1) as a function of J, $n = 1/h$ and ϵ. These results clearly demonstrate that the largest and smallest eigenvalues can be bounded independently of the parameter ϵ for a fixed number of levels. We also note that the largest generalized eigenvalue increases somewhat less than linearly with the number of levels. The smallest eigenvalues decrease with the number of levels but not as fast as suggested by the theory. This may be because we are not yet into the asymptotic range. Overall, the numerically computed eigenvalues are in agreement with the theory of §3.

The corresponding condition numbers can be computed by taking the ratio of the largest to smallest eigenvalue in Table 5.1. For all of the runs reported, the condition numbers are in the interval [8, 56].

TABLE 5.1
Eigenvalues for the hierarchical preconditioner.

		$\epsilon = 10$		$\epsilon = 1$		$\epsilon = .1$	
J	$1/h$	$\lambda_{\max}$	$\lambda_{\min}$	$\lambda_{\max}$	$\lambda_{\min}$	$\lambda_{\max}$	$\lambda_{\min}$
1	8	2.2	.28	2.3	.28	2.4	.28
1	16	2.3	.28	2.4	.28	2.4	.28
1	32	2.4	.28	2.4	.28	2.4	.28
1	64	2.4	.28	2.4	.28	2.4	.28
2	16	3.1	.16	3.2	.17	3.2	.17
2	32	3.2	.17	3.2	.17	3.2	.17
2	64	3.2	.17	3.2	.17	3.2	.17
3	32	3.7	.11	3.8	.11	3.8	.11
3	64	3.8	.11	3.8	.11	3.8	.11
4	64	4.1	.08	4.2	.08	4.2	.08

In the case of multigrid iteration, the rate of convergence is determined by the smallest eigenvalue. If the multigrid process is directly applied, then the reduction is

$$\rho = 1 - \lambda_{\min}.$$

Alternatively, if the multigrid operator is used to accelerate the convergence of conjugate gradient, the average reduction per step is bounded by (5.2) where $\lambda_{\max} = 1$.

We ran into some numerical difficulty computing the lowest eigenvalue for the multigrid process. This is because iterative eigenvalue estimation procedures can converge rather slowly when applied to systems with large condition numbers and possible eigenvalue clustering. We ran the eigenvalue iteration for a thousand iterations on each problem. Although in some cases it was hard to be completely satisfied that the eigenvalue iteration converged, the procedure always provided a valid upper bound for the lowest eigenvalue. Thus, the actual eigenvalue is always smaller than that reported in Table 5.2 and the actual condition number is always worse than the reported estimate.

The numerical results given in Table 5.2 for the symmetric multigrid V-cycle show that the method does not perform well on this application. We see that a decrease in ϵ of a factor of ten leads to roughly a factor of ten increase in the condition number. Similarly, a decrease in mesh size by a factor of two leads to roughly a factor of four increase in the condition number. This is in qualitative agreement with the theoretical estimate (4.8).

TABLE 5.2
The smallest eigenvalue and condition number for the multigrid process.

	$\epsilon = 1$		$\epsilon = .1$		$\epsilon = .01$	
$1/h$	$\lambda_{\max}$	K	$\lambda_{\max}$	K	$\lambda_{\max}$	K
8	1.3×10^{-3}	7.7×10^{2}	1.7×10^{-4}	6.0×10^{3}	1.8×10^{-5}	5.7×10^{4}
16	4.0×10^{-4}	2.5×10^{3}	4.5×10^{-5}	2.2×10^{4}	4.6×10^{-6}	2.2×10^{5}
32	1.1×10^{-4}	9.1×10^{3}	1.2×10^{-5}	8.8×10^{4}	1.2×10^{-6}	8.8×10^{5}

Remark 5.1. We also numerically tested the number of iterations required for a prescribed error reduction using multigrid both as a solver by itself and as a preconditioner for conjugate gradient. Multigrid used by itself failed to converge in a realistic number of iterations. This is consistent with the condition number estimates in Table 5.2. However, when used as a preconditioner, the results were somewhat better. A reasonable rate of reduction (although not as fast as that corresponding to the hierarchical method) was observed in the earlier iterations. This is probably due to some clustering of the eigenvalues.

REFERENCES

[1] R. E. BANK AND T. DUPONT, *An optimal order process for solving finite element equations*, Math. Comp., 36 (1981), pp. 35–51.

[2] M. BERCOVIER, *Perturbation of mixed variational problems—application to mixed finite element methods*, RAIRO, 12 (1978), pp. 211-236.

[3] R. E. BANK, T. F. DUPONT, AND H. YSERANTANT, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[4] 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.

[5] D. BRAESS AND R. VERFURTH, *Multigrid methods for nonconforming finite element methods*, SIAM J. Numer. Anal., 27 (1990), pp. 979–986.

[6] J. H. BRAMBLE AND J. E. PASCIAK, *A preconditioning technique for indefinite systems resulting from mixed approximations of elliptic problems*, Math. Comp., 50 (1988), pp. 1–18.

[7] ———, *New convergence estimates for multigrid algorithms*, Math. Comp., 49 (1987), pp. 311–329.

[8] ———, *The analysis of smoothers for multigrid algorithms*, Math. Comp., 58 (1992), pp. 467–488.

[9] J. H. BRAMBLE, J. E. PASCIAK, J. WANG, AND J. XU, *Convergence estimates for multigrid algorithms without regularity assumptions*, Math. Comp., 57 (1991), pp. 23–45.

[10] J.H. Bramble, J.E. Pasciak, and J. Xu, *The analysis of multigrid algorithms with non-nested spaces or non-inherited quadratic forms*, Math. Comp., 56 (1991), pp. 1–34.

[11] S. Brenner, *A multigrid algorithm for the lowest order Raviart–Thomas mixed triangular finite element method*, SIAM J. Numer. Anal., submitted.

[12] N. Decker, S. Parter, and J. Mandel, *On the role of regularity in multigrid methods*, Multigrid Methods, Proc. of the Third Copper Mountain Conf., S. McCormick, ed., Marcel Dekker, New York, 1988.

[13] R. E. Ewing, R. D. Lazarov, P. Lu, and P. S. Vassilevski, *Preconditioning indefinite systems arising from mixed finite element discretization of second-order elliptic problems*, preprint.

[14] R. E. Ewing, R. D. Lazarov, T. F. Russell, and P. S. Vassilevski, *Local refinement via domain decomposition techniques for mixed finite element methods with rectangular Raviart–Thomas elements*, Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 98–114.

[15] R. Glowinski, W. Kinton, and M. F. Wheeler, *Acceleration of domain decomposition algorithms for mixed finite elements by multi-level methods*, Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 263–289.

[16] R. Glowinski and M. F. Wheeler, *Domain decomposition and mixed finite element methods for elliptic problems*, First International Symposium on 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. 144–172.

[17] C. I. Goldstein, *Multigrid analysis of finite element methods with numerical integration*, Math. Comp., 56 (1991), pp. 409–436.

[18] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer-Verlag, New York, 1985.

[19] M. R. Hanisch, *Multigrid preconditioning for mixed finite element methods*, Ph.D. thesis, Cornell University, Ithaca, NY, 1991.

[20] J. Mandel, S. McCormick, and R. Bank, *Variational multigrid theory*, Multigrid Methods, S. McCormick, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 131–178.

[21] T. P. Mathew, *Domain Decomposition and Iterative Refinement Methods for Mixed Finite Element Discretizations of Elliptic Problems*, Ph.D. thesis, New York University, New York, 1989.

[22] S.F. McCormick, ed., *Multigrid Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[23] P. A. Raviart and J. M. Thomas, *A mixed finite element method for 2-nd order elliptic problems*, Mathematical Aspects of Finite Element Methods, Lecture Notes in Mathematics, #606, I. Galligani and E. Magenes, eds., Springer-Verlag, New York, 1977, pp. 292–315.

[24] R. Temam, *Navier–Stokes Equations*, North-Holland Publishing Co., New York, 1977.

[25] H. Yserentant, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1089–1106, September 1993

ON A MULTIVARIATE EIGENVALUE PROBLEM, PART I: ALGEBRAIC THEORY AND A POWER METHOD*

MOODY T. CHU† AND J. LOREN WATTERSON‡

Abstract. Multivariate eigenvalue problems for symmetric and positive definite matrices arise from multivariate statistics theory where coefficients are to be determined so that the resulting linear combinations of sets of random variables are maximally correlated. By using the method of Lagrange multipliers such an optimization problem can be reduced to the multivariate eigenvalue problem. For over 30 years an iterative method proposed by Horst [*Psychometrika*, 26 (1961), pp. 129–149] has been used for solving the multivariate eigenvalue problem. Yet the theory of convergence has never been complete. The number of solutions to the multivariate eigenvalue problem also remains unknown. This paper contains two new results. By using the degree theory, a closed form on the cardinality of solutions for the multivariate eigenvalue problem is first proved. A convergence property of Horst's method by forming it as a generalization of the so-called power method is then proved. The discussion leads to new formulations of numerical methods.

Key words. multivariate eigenvalue problem, homotopy method, power method

AMS subject classifications. 62H25, 65F30, 65F15, 65H10

1. Introduction.

Given a symmetric and positive definite matrix $A \in R^{n\times n}$ and a set

$$P := \{n_1, \dots, n_m\} \tag{1}$$

of positive integers with $\sum_{i=1}^m n_i = n$, let A be partitioned into blocks

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mm} \end{bmatrix} \tag{2}$$

with $A_{ii} \in R^{n_i \times n_i}$. A *multivariate eigenvalue problem* (MEP) is to find real scalars $\lambda_1, \dots, \lambda_m$ and a real column vector $x \in R^n$ such that equations

$$Ax = \Lambda x, \tag{3}$$

$$\|x_i\| = 1, \qquad i = 1, \dots, m \tag{4}$$

are satisfied, where Λ is the diagonal matrix

$$\Lambda = \operatorname{diag}\{\lambda_1 I^{[n_1]}, \dots, \lambda_m I^{[n_m]}\} \tag{5}$$

and where $I^{[n_i]}$ is the identity matrix of size n_i and $x \in R^n$ is partitioned into blocks

$$x = [x_1^T, \dots, x_m^T]^T, \tag{6}$$

with $x_i \in R^{n_i}$.

*Received by the editors November 25, 1991; accepted for publication (in revised form) September 18, 1992.

†This research was supported in part by the National Science Foundation under grants DMS-9006135 and DMS-9123448.

‡Department of Mathematics, North Carolina State University, Raleigh, North Carolina 27695-8205.

We first note that a multivariate eigenvalue problem is fundamentally different from the so-called multiparameter eigenvalue problem [19]. The latter problem is, given $A_i, B_{is} \in R^{n_i \times n_i}$, to solve the equations

$$\left(A_i - \sum_{s=1}^{m} \lambda_s B_{is}\right) x_i = 0, \qquad i = 1, \ldots, m, \tag{7}$$

for $\lambda = [\lambda_1, \ldots, \lambda_m]^T \in R^m$ and $x_i \in R^{n_i}$.

Equations (3) and (4) represent a nonlinear algebraic system in $n + m$ unknowns. Trivially, if $m = 1$, then (3) is simply a classical symmetric eigenvalue problem. In this case it is well understood that, counting multiplicity, there are exactly n eigenvalues and that, counting negative signs, there are exactly $2n$ eigenvectors if all eigenvalues are distinct. When $m > 1$, however, the concept of characteristic polynomial is no longer applicable. In fact, very little theory is known concerning the characteristic of a solution to the MEP.

The first contribution of this paper is that we are able to determine the cardinality of solutions to the MEP by using degree theory. We prove that a total of $\prod_{i=1}^{m} 2n_i$ solutions exist for the MEP in the generic case. Our method is similar in spirit to that developed in [3], with generalizations. The result apparently is new. As a by-product, homotopy may also be used as a numerical method to find *all* solutions, if so desired, of the MEP.

The MEP has its origins in the determination of canonical correlation coefficients for multivariate statistics [10], [12], [13]. Its history goes back to when Hotelling [10] first studied the so-called maximal correlation problem, which later was developed into what is known as canonical correlation analysis. The numerical method proposed then was somewhat awkward and inefficient. Later, Horst [12] proposed an iterative approach for solving the maximal correlation problem. In spite of a seemingly successful numerical experiment, however, no rigorous proof has ever been developed to show that the iterative procedure converges and that the limit point obtained gives maximal correlation.

The second contribution of this paper is that we reformulate Horst's iterative algorithm as a generalization of the so-called power method [8]. We provide a proof that shows the method does converge monotonically, but only to a *local* maximal correlation. We give an example that shows that Horst's iteration has a good chance of not converging to the *absolute* maximal correlation.

The paper is organized as follows. The statistical background of the MEP is reviewed in §2. Readers who are familiar with the background of the MEP or who are interested only in linear algebra may pass over this section entirely. In §3 we use homotopy theory to prove the cardinality of solutions to the MEP. To accomplish this we establish several auxiliary lemmas that are of interest in their own right. In §4 we reformulate Horst's algorithm and prove the convergence properties. The key of our success lies in the fact that there are only finitely many solutions to the MEP.

Many open questions remain to be studied. The power method can be thought of as a Jacobi-like iterative scheme. Thus other iterative techniques used for linear algebraic equations [9], such as the Gauss–Seidel method or the SOR method, can be modified for solving the MEP. Multivariate shifting is another possible way to find other solutions of the MEP. Some of these issues are briefly mentioned in §5. The details of these new formulations will be discussed in a forthcoming paper [6].

2. Statistical background.

In this section we provide a somewhat detailed statistical background of the MEP. The discussion should also make this paper more self-contained.

However, readers may choose to skip this section entirely without worries of discontinuity. For clarity, the following rule is used in establishing the notation. A calligraphic letter, e.g., $\mathcal{X}$, denotes a scalar random variable, an accented letter, e.g., $\tilde{X}$, denotes an array of random variables, and an uppercase letter, e.g., X or Ω, denotes a matrix.

Given an n-dimensional random variable $\tilde{X} := (\mathcal{X}_1, \ldots, \mathcal{X}_n)$ with a certain distribution function, let $[x_{\xi 1}, \ldots, x_{\xi n}]$, $\xi = 1, \ldots, k$, denote a random sample of size k of this variable. For convenience we use

$$X := [x_{\xi i}] \tag{8}$$

to denote the $k \times n$ sample matrix. Since the notion for a random variable can be carried over in a parallel manner to a random sample and vice versa, in what follows we shall not make a careful distinction between a random variable $\mathcal{X}_i$ and its corresponding random sample $[x_{1i}, \ldots, x_{ki}]^T$.

By shifting if necessary, we may assume without loss of generality that the sample mean $\mu_i := \sum_{\xi=1}^{k} \frac{x_{\xi i}}{k}$ for each of the random variable $\mathcal{X}_i$ is zero. It follows then that the $n \times n$ matrix

$$\Delta := X^T X \tag{9}$$

represents the covariance matrix of the random sample X. Clearly, Δ is symmetric. We further assume as a generic case that there is no degenerate component and no linear dependence among the components $\mathcal{X}_1, \ldots, \mathcal{X}_n$. It is well known that Δ is positive definite [22].

Corresponding to the same conformation as in (6), let the components of $\tilde{X}$ be divided into mutually disjoint groups

$$\tilde{X} = (\tilde{X}_1, \ldots, \tilde{X}_m). \tag{10}$$

Then each $\tilde{X}_i$ is an n_i-dimensional random variable. Let the matrix

$$\Delta = [\Delta_{ij}] \tag{11}$$

be partitioned in the same way as (2). Then Δ_{ii} represents the covariance matrix of the $k \times n_i$ sample block X_i, where the sample matrix X is partitioned as

$$X = [X_1, \ldots, X_m]. \tag{12}$$

In practice it is often desirable to simplify the analysis by combining all n_i components of $\tilde{X}_i$ linearly into a single new variable $\mathcal{Z}_i$. For $i = 1, \ldots, m$ let $b_i \in R^{n_i}$ denote the coefficients of linear combinations for variable $\tilde{X}_i$. Define the $n \times m$ matrix

$$B := \begin{bmatrix} b_1 & 0 & \cdots & 0 \\ 0 & b_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & b_m \end{bmatrix}. \tag{13}$$

Then the sample matrix X is transformed into the $n \times m$ matrix

$$Z := XB := [Z_1, \ldots, Z_m]. \tag{14}$$

The covariance matrix corresponding to the new random sample Z is given by

$$\Omega := Z^T Z = B^T \Delta B. \tag{15}$$

Consider the case $m = 2$ as an example. It is often desirable to use $\mathcal{Z}_1$ to predict $\mathcal{Z}_2$. Thus it is imperative to find b_1 and b_2 so that the correlation coefficient between the two new random samples Z_1 and Z_2 is as large as possible. The covariance matrix of Z is seen to be the 2×2 matrix

$$\Omega = \begin{bmatrix} b_1^T \Delta_{11} b_1 & b_1^T \Delta_{12} b_2 \\ b_2^T \Delta_{21} b_1 & b_2^T \Delta_{22} b_2 \end{bmatrix}. \tag{16}$$

The correlation coefficient ρ to be maximized is

$$\rho = \frac{b_1^T \Delta_{12} b_2}{\sqrt{b_1^T \Delta_{11} b_1}\sqrt{b_2^T \Delta_{22} b_2}}. \tag{17}$$

If the variances of Z_1 and Z_2 are normalized to unity, then to maximize ρ is equivalent to

$$\text{maximize} \quad b^T \Delta b \tag{18}$$

$$\text{subject to} \quad b_i^T \Delta_{ii} b_i = 1 \quad \text{for } i = 1, 2, \tag{19}$$

where

$$b := [b_1^T, b_2^T]^T. \tag{20}$$

Since each Δ_{ii} is symmetric and positive definite, the Cholesky decomposition

$$\Delta_{ii} = T_i^T T_i \tag{21}$$

exists. Introduce the block diagonal matrix

$$T := \operatorname{diag}\{T_1, T_2\}, \tag{22}$$

and define

$$x := Tb := [x_1^T, x_2^T]^T, \tag{23}$$

$$A := T^{-T} \Delta T^{-1}, \tag{24}$$

where $T^{-T} = {T^{-1}}^T$. Clearly, A is still symmetric and positive definite. The problem (18) and (19) is now transformed into

$$\text{maximize} \quad x^T A x \tag{25}$$

$$\text{subject to} \quad x_i^T x_i = 1 \quad \text{for } i = 1, 2. \tag{26}$$

Using the method of Lagrange multipliers [17], we now form the Lagrangian function

$$\phi(x, \lambda_1, \lambda_2) := x^T A x - \sum_{i=1}^{2} \lambda_i (x_i^T x_i - 1), \tag{27}$$

with λ_1 and λ_2 as the Lagrange multipliers. Upon differentiating (27) it is clear that the maximal correlation problem (18) and (19) for $m = 2$ is reduced to the 2-variate eigenvalue problem (3) and (4).

When $m > 2$ the requirement of maximizing the correlation coefficients among the m random samples $Z_1, \ldots, Z_m$ needs to be modified. Intuitively, the more *similar* the vectors $Z_1, \ldots, Z_m$ are to each other, the more closely the correlation coefficients will approach unity. Thus it makes sense to require that the sum of the off-diagonal elements of $m \times m$ matrix Ω in (15) be maximized subject to the condition that the diagonal elements of Ω be unity. Apparently, the maximal correlation problem for the $m > 2$ case can now be formulated in the same way as (18) and (19). We may repeat the same procedure as for the case $m = 2$ to argue that a solution to the maximal correlation problem

$$\text{(28)} \qquad \text{maximize} \quad x^T A x$$

$$\text{(29)} \qquad \text{subject to} \quad x_i^T x_i = 1 \quad \text{for } i = 1, \ldots, m$$

is necessarily a solution to the multivariate eigenvalue problem.

Suppose that a set of solutions $\{b_1^{(1)}, \ldots, b_m^{(1)}\}$ to the maximal correlation problem has been determined. We may wish to find yet another set of solutions $\{b_1^{(2)}, \ldots, b_m^{(2)}\}$ so that the resulting composites $Z_1^{(2)}, \ldots, Z_m^{(2)}$ are also maximally correlated. Since the second measurement is conducted independently of the first, it is natural to require that variables within the same class are uncorrelated, that is, each $Z_i^{(2)}$ is correlated zero with the corresponding $Z_i^{(1)}$. This procedure may be repeated until we have obtained $p := \min\{n_1, \ldots, n_m\}$ sets of solutions. The zero correlation is required between any two variables in the class $\{Z_i^{(1)}, \ldots, Z_i^{(p)}\}$. The reason that such a repeated measurement is needed can be found from many practical applications [10], [12], [13]. We note that variables from different classes are not subject to any correlation restrictions, although ideally we would like these variables to be zero correlated as well.

The new problem mentioned above was originally studied by Hotelling [10], who mainly focused on the case in which $m = 2$, but the numerical method proposed was very inefficient. Horst [12] then developed a direct approach (for $m = 2$), which we found to use techniques of what is now known as singular-value decomposition. It is worthwhile to rewrite the direct method in terms of the current notion as follows.

We first generalize the notation B in (13) to be the $n \times 2p$ matrix

$$\text{(30)} \qquad B := \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} := \begin{bmatrix} b_1^{(1)} & \cdots & b_1^{(p)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_2^{(1)} & \cdots & b_2^{(p)} \end{bmatrix}.$$

The random sample Z then becomes an $n \times 2p$ matrix

$$\text{(31)} \qquad Z := XB := [Z_1^{(1)}, \ldots, Z_1^{(p)}, Z_2^{(1)}, \ldots, Z_2^{(p)}].$$

Consider the $2p \times 2p$ covariance matrix $\Omega = B^T \Delta B$ of Z. Recall that each Δ_{ii} has a Cholesky decomposition (21). Let

$$\text{(32)} \qquad T_1^{-T} \Delta_{12} T_2^{-1} = Q_1^T D Q_2$$

be the singular-value decomposition [8] of the $n_1 \times n_2$ matrix $T_1^{-T}\Delta_{12}T_2^{-1}$. It follows that

$$(33)\quad \Omega = \begin{bmatrix} B_1^T T_1^T Q_1^T & 0 \\ 0 & B_2^T T_2^T Q_2^T \end{bmatrix} \begin{bmatrix} I^{[n_1]} & D \\ D & I^{[n_2]} \end{bmatrix} \begin{bmatrix} Q_1 T_1 B_1 & 0 \\ 0 & Q_2 T_2 B_2 \end{bmatrix}.$$

If we choose the vectors in B_1 and B_2 to be

$$(34)\qquad B_1 = T_1^{-1} Q_1^T I_{n_1 \times p},$$

$$(35)\qquad B_2 = T_2^{-1} Q_2^T I_{n_2 \times p},$$

where $I_{s \times t}$ means the first $s \times t$ submatrix of the identity, then

$$(36)\qquad \Omega = \begin{bmatrix} I^{[p]} & \Sigma \\ \Sigma & I^{[p]} \end{bmatrix},$$

where Σ is the diagonal matrix of singular values $\sigma_1 \geq \cdots \geq \sigma_p \geq 0$. In other words, if (34) and (35) are satisfied, then σ_i is precisely the highest correlation coefficient between $Z_1^{(i)}$ and $Z_2^{(i)}$, and each of these two variables is uncorrelated to any other $Z_s^{(t)}$ for $s =$ 1 or 2 and $t \neq i$.

For $m > 2$ the coefficient matrix B is generalized to a block diagonal matrix

$$(37)\qquad B := \operatorname{diag}\{B_1, \ldots, B_m\},$$

where each B_i is an $n_i \times p$ matrix

$$(38)\qquad B_i := \left[b_i^{(1)}, \ldots, b_i^{(p)}\right]$$

and the random sample becomes

$$(39)\qquad Z := XB := [Z_1^{(1)}, \ldots, Z_1^{(p)}, \ldots, Z_m^{(1)}, \ldots, Z_m^{(p)}].$$

Unfortunately, Horst's direct method cannot be generalized to solve the general m case. The reason is simply that there is no way to transform all $m \times m$ blocks of size $p \times p$ in

$$(40)\qquad \Omega = [B_i^T \Delta_{ij} B_j]$$

into diagonal matrices simultaneously when $m > 2$. To remedy this, it was at this point that Horst introduced his iterative method [12], [13] without a proof.

3. Homotopy method and cardinality.

In this section we use degree theory to prove the cardinality of solutions to the MEP. The theory has been used as a major tool in analysis to prove the existence of solutions for a wide variety of problems. For a complete mathematical treatment of the theory, we refer the reader to [16]. Degree theory can often be implemented as a numerical means, known as the homotopy method, of computing a solution of a nonlinear system. Applications of homotopy methods can be found, for example, in [1]–[5], [7], [15], [18], [20], [21].

Rewrite the multivariate eigenvalue problem as a nonlinear system:

$$(41)\qquad F(x, \Lambda) = 0,$$

where $F : R^n \times R^m \longrightarrow R^n \times R^m$ is defined by

$$F(x, \Lambda) := \begin{bmatrix} \Lambda x - Ax \\ \frac{x_1^T x_1 - 1}{2} \\ \vdots \\ \frac{x_m^T x_m - 1}{2} \end{bmatrix}. \tag{42}$$

In general, it is not an easy task to find a solution for (41). It seems perhaps even harder to count the total number of solutions. On the other hand, consider a simple MEP:

$$\begin{aligned} Dx &= \Lambda x, \\ \|x_i\| &= 1, \qquad i = 1, \ldots, m, \end{aligned} \tag{43}$$

where D is a diagonal matrix with distinct elements $d_1^{(1)}, \ldots, d_{n_1}^{(1)}, \ldots, d_1^{(m)}, \ldots, d_{n_m}^{(m)}$. It is trivial to see the following lemma.

LEMMA 3.1. *The problem* (43) *has exactly* $\prod_{i=1}^m 2n_i$ *solutions. These are, for* $i = 1, \ldots, m$,

$$\begin{aligned} \lambda_i &= d_{j_i}^{(i)}, \\ x_i &= \pm e_{j_i}^{[n_i]}, \end{aligned} \tag{44}$$

where $j_i = 1, \ldots, n_i$ *and* $e_s^{[t]}$ *denotes the sth column of the identity matrix* $I^{[t]}$.

Our basic idea is to construct a homotopy between (43) and (3) so that no homotopy curve will escape to infinity or turn back. Toward this end we define a function $H : R^n \times R^m \times R \longrightarrow R^n \times R^m$ as follows:

$$H(x, \Lambda, t; D) := \begin{bmatrix} \Lambda x - [D + t(A - D)]x \\ \frac{x_1^T x_1 - 1}{2} \\ \vdots \\ \frac{x_m^T x_m - 1}{2} \end{bmatrix}, \tag{45}$$

where D is a diagonal matrix whose elements will be specified later. The main concern is to show that the Jacobian $D_{x,\Lambda,t}H$ of H is of full rank if D is appropriately chosen. In what follows we establish several auxiliary lemmas to help this investigation. The lemmas, which themselves are of interest, concern the spectrum property when a simple matrix M is perturbed by a diagonal matrix.

A matrix M is simple if and only if the algebraic multiplicities and geometric multiplicities of each of its eigenvalues coincide. Symmetric matrices are automatically simple. We begin with a fact that follows from a more general result in [11, Thm. 1.4.9].

LEMMA 3.2. *Suppose that* M *is a simple matrix in* $R^{n \times n}$ *and that* $r \geq 1$ *is a positive integer. Then* λ *is an eigenvalue of* M *of multiplicity* r *if and only if* λ *is an eigenvalue of all* $q \times q$ *principal submatrices of* M *whenever* $q \geq n - r + 1$.

For convenience we denote the diagonally perturbed matrix by

$$\mathcal{M} = \mathcal{M}(D) := M + D, \tag{46}$$

with $D = \operatorname{diag}\{d_1, \ldots, d_n\}$. We claim the following.

LEMMA 3.3. *Let $E := \{(d_1, \ldots, d_n) \in R^n | \mathcal{M}$ has multiple eigenvalues$\}$. Then the complement U of E is open, is dense, and has full Lebesgue measure in R^n.*

Proof. Let

$$E_r := \{(d_1, \ldots, d_n) \in R^n | \mathcal{M} \text{ has zero eigenvalue of multiplicity } r\}. \tag{47}$$

By Lemma 3.2 there exists an $(n - r) \times (n - r)$ principal submatrix $\hat{\mathcal{M}}$ of $\mathcal{M}$ such that $\hat{\mathcal{M}}$ is nonsingular. Without loss of generality we may assume $\hat{\mathcal{M}}$ is the leading principal submatrix of $\mathcal{M}$, i.e., $\hat{\mathcal{M}}$ is indexed by $\{1, \ldots, n - r\}$. All principal submatrices of $\mathcal{M}$ with size $\geq n - r + 1$ are singular. In particular, for $i = 1, \ldots, r$ the principal submatrix $\mathcal{M}_i$ indexed by $\{1, \ldots, n - r, n - r + i\}$ is singular.

Define

$$f_i(d_1, \ldots, d_n) := \det(\mathcal{M}_i). \tag{48}$$

Then the value of $(d_1, \ldots, d_n)$ that causes $\mathcal{M}$ to have rank $n - r$ must satisfy the system of equations

$$\mathcal{F}(d_1, \ldots, d_n) = 0, \tag{49}$$

where $\mathcal{F} : R^n \longrightarrow R^r$ is defined by $\mathcal{F} := (f_1, \ldots, f_r)$.

Observe that

$$\frac{\partial f_i}{\partial d_{n-r+i}} = \det(\hat{\mathcal{M}}) \neq 0, \tag{50}$$

and hence

$$\frac{\partial \mathcal{F}}{\partial(d_{n-r+1}, \ldots, d_n)} = \det(\hat{\mathcal{M}}) I^{[r]} \tag{51}$$

is nonsingular. By the implicit function theorem we thus know that $\{d_{n-r+1}, \ldots, d_n\}$ can be rewritten as functions of $\{d_1, \ldots, d_{n-r}\}$. Thus E_r is necessarily embedded in a manifold of dimension $n - r$.

Let $e := [1, \ldots, 1] \in R^n$. Obviously, $\langle e \rangle \oplus E_r$ contains all values of $(d_1, \ldots, d_n)$ such that $M + D$ has an eigenvalue of multiplicity r. From the fact that

$$E = \langle e \rangle \oplus \bigcup_{r=2}^{n} E_r \tag{52}$$

we see that U is open, is dense, and has full Lebesgue measure in R^n. $\square$

By using the resultant theorem a complex version of the above lemma has been proved in [5, Thm. 2.3]. The new contribution here is that Lemma 3.3 is for real simple matrices perturbed by real diagonal matrices.

We note that (49) is only a necessary condition for $\mathcal{M}$ being rank deficient by r. Conceivably, the set E_r could be much smaller than what the dimension $n - r$ suggests. This is especially so when M is a symmetric matrix since we have the following observation.

Rewrite $\mathcal{M}$ as

$$\mathcal{M} = \begin{bmatrix} \hat{\mathcal{M}} & \hat{\mathcal{P}} \\ \hat{\mathcal{P}}^T & \hat{\mathcal{R}} \end{bmatrix}. \tag{53}$$

Let

$$\hat{\mathcal{L}} = \begin{bmatrix} I^{[n-r]} & 0 \\ -(\hat{\mathcal{M}}^{-1}\hat{\mathcal{P}})^T & I^{[r]} \end{bmatrix}. \tag{54}$$

Then the rank condition of $\mathcal{M}$ is the same as that of

$$\mathcal{N} := \hat{\mathcal{L}}\mathcal{M}\hat{\mathcal{L}}^T = \begin{bmatrix} \hat{\mathcal{M}} & 0 \\ 0 & \hat{\mathcal{R}} - \hat{\mathcal{P}}^T\hat{\mathcal{M}}^{-1}\hat{\mathcal{P}} \end{bmatrix}. \tag{55}$$

In particular, if $\mathcal{M}$ is of rank $n-r$, then the lower-right $r \times r$ block of $\mathcal{N}$ must be identically zero, which gives rise to $r(r+1)/2$ equations in the $d_1, \ldots, d_n$.

The r diagonal elements of $\hat{\mathcal{R}} - \hat{\mathcal{P}}^T\hat{\mathcal{M}}^{-1}\hat{\mathcal{P}}$ are readily solvable for $d_{n-r+1}, \ldots, d_n$ in terms of $d_1, \ldots, d_{n-r}$, as is predicted by the implicit function theorem in the proof of Lemma 3.3. The off-diagonal elements impose $r(r-1)/2$ extra conditions that $d_1, \ldots, d_{n-r}$ must satisfy. For $r = 2$ it is easy to see that this extra condition is not a trivial equation for $d_1, \ldots, d_{n-2}$. Therefore, E_2 should be a manifold of dimension $n-3$ (rather than $n-2$). It should be rather obvious (although tedious to prove) that E_r is generally a manifold of codimension $r(r+1)/2$ since there are a total of $r(r+1)/2$ equations that $d_1, \ldots, d_n$ must satisfy. Obviously, E_r is empty if r is too large. Indeed, by the Wilson–Ledermann bound [14], E_r is empty if

$$r > \left\lceil \frac{2n+1-\sqrt{8n+1}}{2} \right\rceil. \tag{56}$$

We do not intend to provide here a proof for the exact dimension of each E_r. For the purpose of this paper it suffices to see from the above argument that the following lemma holds.

LEMMA 3.4. *For any generic and symmetric matrix M,* $\dim(E_r) \leq n-3$ *for $r \geq 2$ and hence* $\dim(E) \leq n-2$.

We demonstrate the case $n = 4$ and $r = 2$ as an example. Suppose M is denoted as $M := [m_{ij}]$. The extra equation to be satisfied by d_1 and d_2 is

$$m_{34} = \frac{m_{13}m_{22}m_{14}+m_{14}m_{13}d_2-m_{12}m_{14}m_{23}+d_1m_{24}m_{23}-m_{12}m_{13}m_{24}+m_{24}m_{23}m_{11}}{m_{11}m_{22}+m_{11}d_2+d_1m_{22}+d_1d_2-m_{12}^2},$$

and E_2 is made of points (d_1, d_2, d_3, d_4), where

$$d_2 = \frac{d_1m_{24}m_{23}-d_1m_{22}m_{34}-m_{22}m_{34}m_{11}+m_{13}m_{22}m_{14}+m_{34}m_{12}^2+m_{24}m_{23}m_{11}-m_{12}m_{14}m_{23}-m_{12}m_{13}m_{24}}{d_1m_{34}+m_{34}m_{11}-m_{14}m_{13}},$$

$$d_3 = \frac{d_1m_{23}m_{34}-d_1m_{33}m_{24}+m_{11}m_{23}m_{34}-m_{13}m_{14}m_{23}+m_{24}m_{13}^2-m_{12}m_{13}m_{34}+m_{12}m_{33}m_{14}-m_{11}m_{33}m_{24}}{d_1m_{24}+m_{24}m_{11}-m_{14}m_{12}},$$

$$d_4 = \frac{d_1m_{24}m_{34}-d_1m_{23}m_{44}+m_{11}m_{24}m_{34}-m_{14}m_{13}m_{24}-m_{12}m_{14}m_{34}-m_{11}m_{23}m_{44}+m_{23}m_{14}^2+m_{12}m_{13}m_{44}}{d_1m_{23}-m_{13}m_{12}+m_{23}m_{11}}.$$

Obviously, E_2 is a 1-dimensional manifold parameterized in d_1.

We are now ready to establish one of the major results.

LEMMA 3.5. *The set of D such that the matrix $\Lambda - (D + t(A-D))$ is of rank less than $n-m$ for some Λ and some $t \in (0,1)$ is of measure zero.*

Proof. For convenience we denote

$$\mathcal{A} = \mathcal{A}(\Lambda, t, D) := \Lambda - (D + t(A - D)). \tag{57}$$

Observe that each of the m diagonal blocks of $\mathcal{A}$ takes the form

$$\mathcal{A}_{ii} = \lambda_i I^{[n_i]} - (1-t)\text{diag}(d_1^{(i)}, \ldots, d_{n_i}^{(i)}) - tA_{ii}. \tag{58}$$

If the rank of $\mathcal{A}$ is less than $n - m$, then one of the diagonal blocks of $\mathcal{A}$ must be rank deficient by at least two. Equivalently, this implies that for some $\tau \in (0, \infty)$ the matrix τ diag $(d_1^{(i)}, \ldots, d_{n_i}^{(i)}) + A_{ii}$ has an eigenvalue with multiplicity at least two. By Lemma 3.4 the union over τ

$$\bigcup_{\tau \in (0,\infty)} \{(d_1^{(i)}, \ldots, d_{n_i}^{(i)}) | \tau \text{ diag } (d_1^{(i)}, \ldots, d_{n_i}^{(i)}) + A_{ii} \text{ has multiple eigenvalues}\}$$

is of dimension at most $n_i - 1$. Thus, overall, the set of D such that $\Lambda - (D + t(A - D))$ is of rank less than $n - m$ for some Λ and some $t \in (0, 1)$ is at most of geometric dimensional $n - 1$. $\square$

Henceforth we shall assume that D in (45) has been chosen so that $\mathcal{A}(\Lambda, t, D)$ has rank at least $n - m$ for all Λ and all t. We now prove the existence of the homotopy curves.

LEMMA 3.6. *The point* $0 \in R^n \times R^m$ *is a regular value for* H. *That is, for each* $(x, \Lambda, t) \in R^n \times R^m \times R$ *such that* $H(x, \Lambda, t) = 0$, *the Jacobian matrix* $D_{(x,\Lambda,t)}H$ *has rank* $n + m$.

Proof. For convenience we divide the Jacobian matrix $D_{(x,\Lambda,t)}H$ into blocks:

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} & \mathcal{C} \\ \mathcal{B}^T & 0 & 0 \end{bmatrix}, \tag{59}$$

where $\mathcal{A}$ is as given in (57),

$$\mathcal{B} := \begin{bmatrix} x_1 & 0 & \ldots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & x_m \end{bmatrix}, \tag{60}$$

and

$$\mathcal{C} := (D - A)x. \tag{61}$$

If $H(x, \Lambda, t) = 0$, then

$$\mathcal{A}x = 0, \tag{62}$$

which indicates that x is an eigenvector of $\mathcal{A}$ with corresponding eigenvalue zero.

Suppose that for a certain $1 \leq i \leq m$ there exists $y_i \in R^n$ such that

$$\mathcal{A}y_i = \begin{bmatrix} 0 \\ \vdots \\ x_i \\ 0 \\ \vdots \end{bmatrix}. \tag{63}$$

Then, by the symmetry of $\mathcal{A}$, we know

$$\langle \mathcal{A}y_i, x\rangle = \langle y_i, \mathcal{A}x\rangle = 0. \tag{64}$$

On the other hand, we should have

$$\langle \mathcal{A}y_i, x\rangle = \langle x_i, x\rangle = 1. \tag{65}$$

This contradiction implies that none of the m columns of $\mathcal{B}$ can be in the range of $\mathcal{A}$.

Since $\mathcal{A}$ is at least of rank $n-m$ by Lemma 3.5, it is now clear that the $n \times (n+m)$ matrix $[\mathcal{A}, \mathcal{B}]$ is of rank n. It is also clear the rows of $[\mathcal{B}^T, 0]$ are not in the row space of $[\mathcal{A}, \mathcal{B}]$. Thus the assertion is proved. □

By now the following theorem is a standard result from the differential topology.

THEOREM 3.7. *The set* $\Gamma := \{(x, \Lambda, t) | H(x, \Lambda, t) = 0\}$ *is a* 1*-dimensional smooth submanifold in* $R^n \times R^m \times R$.

Furthermore, since $D_{(x,\Lambda,t)}H$ is nonsingular, the implicit function theorem asserts that each component of Γ can be characterized as a function of t. Also, if $(x, \Lambda, t) \in \Gamma$, then

$$\sum_{i=1}^{m} \lambda_i^2 = ||\Lambda x|| = ||((1-t)D + tA)x|| \leq m(||(1-t)D|| + ||tA||), \tag{66}$$

which implies that no homotopy curve can diverge to infinity for $t \in (0,1)$.

Putting all the above arguments together, we have proved that for $i = 1, \dots, m$, the solution to the initial-value problem

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ \vdots \\ x_m \\ \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix} = \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{B}^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} -\mathcal{C} \\ 0 \end{bmatrix}, \tag{67}$$

$$x_i(0) = \pm e_{j_i}^{[n_i]}, \tag{68}$$

$$\lambda_i(0) = d_{j_i}^{(i)} \tag{69}$$

is a curve in $R^n \times R^m$ that extends from $t = 0$ to $t = 1$. From Lemma 3.1 we thus conclude the following.

THEOREM 3.8. *For a generic and symmetric matrix A the MEP has exactly $\prod_{i=1}^{m} 2n_i$ solutions.*

We believe the result of Theorem 3.8 is new. Moreover, it should be pointed out that the positive definiteness of the matrix A is not needed in the proof of Theorem 3.8.

4. Power method and convergence.

Horst's algorithm may be reformulated as follows.

ALGORITHM 4.1 (Horst's algorithm).

Given $x^{(0)} = (x_1^{(0)^T}, \dots, x_m^{(0)^T})^T$ *with* $||x_i^{(0)}|| = 1$, *do*

```
for $k = 1, 2, \dots$
    for $i = 1, \dots, m$
```

$$y_i^{(k)} := \sum_{j=1}^{m} A_{ij} x_j^{(k)}, \tag{70}$$

$$\lambda_i^{(k)} := \|y_i^{(k)}\|, \tag{71}$$

$$x_i^{(k+1)} := \frac{y_i^{(k)}}{\lambda_i^{(k)}}. \tag{72}$$

```
    end
end
```

Theoretically, it is possible that $\|y_i^{(k)}\| = 0$ and hence that (72) is not well defined. Such a breakdown, although rarely occurring in practice, can easily be remedied by redefining $x_i^{(k+1)}$ to be an arbitrary unit vector in R^{n_i}, say, $x_i^{(k)}$ itself, and then continuing the iteration.

To our knowledge, Horst's iterative algorithm has never been rigorously proved to converge, although some intuitive support as well as some numerical evidence are mentioned in [12], [13]. In this section we prove a convergence property of Algorithm 4.1 for the MEP.

We find that it is more convenient to write the above algorithm in the compact form

$$Ax^{(k)} = \Lambda^{(k)} x^{(k+1)}, \tag{73}$$

where

$$x^{(k)} := [x_1^{(k)^T}, \dots, x_m^{(k)^T}]^T \tag{74}$$

and

$$\Lambda^{(k)} := \operatorname{diag}\{\lambda_1^{(k)} I^{[n_1]}, \dots, \lambda_m^{(k)} I^{[n_m]}\}. \tag{75}$$

The iterative scheme may be viewed as a generalization of the classical power method. The convergence property of this method, nonetheless, is not nearly obvious. As an example, without the positive definiteness the method may fail to converge. This can be seen from the 2×2 matrix $A = \begin{bmatrix} 1 & b \\ b & c \end{bmatrix}$ with $m = 2$. There are exactly four feasible solutions $(\pm 1, \pm 1)$. For any values of b and c satisfying $|b| < 1$, $c < 0$, and $|b| < -c$, the matrix A is not positive definite. In this case the iterations alternate between $(1, 1)$ and $(1, -1)$ or between $(-1, -1)$ and $(-1, 1)$. No convergence occurs. As another example that will be illustrated later, a limit point of the method may depend on the starting point. The maximal correlation problem may have multiple local solutions even if the matrix A is positive definite.

To study the convergence property of Horst's algorithm for the MEP, we denote the objective function in (28) by

$$r(x) := x^T A x \tag{76}$$

for $x \in R^n$. We claim the following theorem.

THEOREM 4.2. *Suppose A is symmetric and positive definite. Then $\{r(x^{(k)})\}$ with $x^{(k)}$ generated by (73) is a monotonically increasing sequence and converges.*

Proof. From (73) we know that

$$r(x^{(k)}) = x^{(k+1)^T}\Lambda^{(k)}x^{(k)}. \tag{77}$$

We also know that

$$r(x^{(k+1)}) = x^{(k+1)^T}Ax^{(k+1)} + x^{(k+1)^T}\Lambda^{(k)}x^{(k+1)} - x^{(k+1)^T}Ax^{(k)}. \tag{78}$$

Subtracting (77) from (78), we obtain

$$r(x^{(k+1)}) - r(x^{(k)}) = x^{(k+1)^T}\Lambda^{(k)}(x^{(k+1)} - x^{(k)}) + x^{(k+1)^T}A(x^{(k+1)} - x^{(k)}). \tag{79}$$

On the other hand, observe that

$$\begin{aligned} & x^{(k)^T}\Lambda^{(k)}(x^{(k+1)} - x^{(k)}) + x^{(k)^T}A(x^{(k+1)} - x^{(k)}) \\ &= x^{(k)^T}Ax^{(k)} - x^{(k)^T}\Lambda^{(k)}x^{(k)} + x^{(k+1)^T}\Lambda^{(k)}x^{(k+1)} - x^{(k)^T}Ax^{(k)} \\ &= 0 \end{aligned} \tag{80}$$

because $x^{(k)^T}\Lambda^{(k)}x^{(k)} = \sum_{i=1}^m \lambda_i^{(k)}$. Subtracting (80) from the right-hand side of (79), we find that

$$r(x^{(k+1)}) - r(x^{(k)}) = (x^{(k+1)} - x^{(k)})^T(A + \Lambda^{(k)})(x^{(k+1)} - x^{(k)}). \tag{81}$$

Since $\Lambda^{(k)}$ is a diagonal matrix with positive elements, the matrix $A + \Lambda^{(k)}$ remains to be symmetric and positive. It follows that

$$r(x^{(k+1)}) \geq r(x^{(k)}). \tag{82}$$

The convergence of the monotone sequence $\{r(x^{(k)})\}$ is obvious since for all x we have

$$\mu_n\|x\|^2 \leq r(x) \leq \mu_1\|x\|^2, \tag{83}$$

where

$$\mu_1 \geq \cdots \geq \mu_n \tag{84}$$

are the eigenvalues of A. □

It is instructive to motivate Theorem 4.2 from another viewpoint. Let $S_i := \{x_i | x_i \in R^{n_i},\ \|x_i\| = 1\}$. The feasible set (29) for the maximal correlation problem may be regarded as the manifold $\prod_{i=1}^m S_i$ embedded in $\prod_{i=1}^m R^{n_i}$ with product topology. The tangent space $\mathcal{T}_{(x_1,\ldots,x_m)}\prod_{i=1}^m S_i$, therefore, is given by $\prod_{i=1}^m \mathcal{T}_{x_i}S_i$. It is easy to show that the gradient $\nabla r(x)$ projected onto $\mathcal{T}_{x_i}S(i)$ is twice the vector

$$g_i(x) := \sum_{j=1}^m A_{ij}x_j - \left\langle x_i, \sum_{j=1}^m A_{ij}x_j \right\rangle x_i. \tag{85}$$

Observe then that

$$
\begin{aligned}
\langle x^{(k+1)} - x^{(k)}, g(x^{(k)})\rangle &= \sum_{i=1}^{m} \langle x_i^{(k+1)} - x_i^{(k)}, g_i(x^{(k)})\rangle \\
&= \sum_{i=1}^{m} \left(\|y_i^{(k)}\| - \frac{\langle x_i^{(k)}, y_i^{(k)}\rangle^2}{\|y_i^{(k)}\|} \right),
\end{aligned} \tag{86}
$$

while each term in (86) is nonnegative. That is, the vector $x^{(k+1)} - x^{(k)}$ forms an acute angle with the projected gradient $g(x^{(k)})$ of $r(x^{(k)})$ and thus probably (although not necessarily) points to an ascent direction for $r(x)$.

Define the residual

$$
\begin{aligned}
\delta(x^{(k)}) &:= Ax^{(k)} - \Lambda^{(k)} x^{(k)} \\
&= \Lambda^{(k)}(x^{(k+1)} - x^{(k)}),
\end{aligned} \tag{87}
$$

where the second equality follows from (73). If the equality in (82) holds for some k, then from (81) it must be $x^{(k+1)} = x^{(k)}$. We find from (87) that $x^{(k)}$ solves (3) exactly. In general, we have the following theorem.

THEOREM 4.3. *The residual* $\{\delta(x^{(k)})\}$ *in Horst's algorithm converges to zero as* $k \longrightarrow \infty$.

Proof. By the definition (71) all $\lambda_i^{(k)}$ remain bounded as $\|x^{(k)}\|^2 \le m$ for all k. It follows from (81) that there exists a constant $\kappa > 0$ such that

$$
r(x^{(k+1)}) - r(x^{(k)}) \ge \kappa \|x^{(k+1)} - x^{(k)}\|^2 \tag{88}
$$

for all k. The assertion follows from (87) and the fact that $\{r(x^{(k)})\}$ converges. □

At this point it is not obvious that the sequence $\{x^{(k)}\}$ itself converges, but it is clear that the sequence $\{x^{(k)}\}$ does have have cluster point(s) due to its boundedness. From Theorem 4.3, it follows that every cluster point x^* satisfies (3) with eigenvalues $\lambda_i^* := \|\sum_{j=1}^{m} A_{ij} x_j^*\|$.

Define

$$
\tilde{\lambda}^{(k)} := \min_{1 \le j \le m} \lambda_j^{(k)} \tag{89}
$$

and

$$
\tilde{\lambda} := \liminf \tilde{\lambda}^{(k)}. \tag{90}
$$

Consider the case in which $\tilde{\lambda} = 0$. Then there exists a subsequence $\{k_j\}$ of positive integers such that

$$
\tilde{\lambda}^{(k_j)} < \frac{1}{j}. \tag{91}
$$

Any convergent subsequence of the subsequence $\{x^{(k_j)}\}$ must be such that one of the eigenvalues λ_i^* is zero. We do not think this will happen often for a generic A.

The following lemma from real analysis is useful.

LEMMA 4.4. *Let $\{a_k\}$ be a bounded sequence of real numbers with the property $|a_{k+1} - a_k| \longrightarrow 0$ as $k \longrightarrow \infty$. If there are only finitely many limit points for the sequence, then $\{a_k\}$ converges to a unique limit point.*

Proof. Suppose $\{a_{\alpha_k}\}$ and $\{a_{\beta_k}\}$ are two subsequences of $\{a_k\}$ that converge, respectively, to two distinct limit points, x and y. Let z denote any fixed real number between x and y. For a positive number r let $B_x(r)$ denote the neighborhood $[x-r, x+r]$ of x.

For any $\epsilon > 0$ that is less than $\frac{1}{4}\min\{|x-z|, |y-z|\}$, there exists a large enough integer $K = K(\epsilon)$ such that

$$a_{\alpha_k} \in B_x(\epsilon), \quad a_{\beta_k} \in B_y(\epsilon), \quad |a_{k+1} - a_k| < \epsilon$$

for all $k \geq K$. Infinitely many elements of $\{a_k\}$ must leave $B_x(\epsilon)$ to enter $B_y(\epsilon)$ and vice versa. Thus there exists an index $\gamma \geq K$ such that $a_\gamma \in B_z(\epsilon)$. This shows that z is also a limit point.

Since z is arbitrary, we have shown that any number between x and y is a limit point. This contradicts the assumption that there are only finitely many limit points. $\square$

In the following we prove that Algorithm 4.1 generates a convergent sequence $\{\Lambda^{(k)}, x^{(k)}\}$. We believe this result is new.

First we prove the following theorems.

THEOREM 4.5. *The sequence $\{\Lambda^{(k)}\}$ converges as k goes to infinity.*

Proof. We have seen already that the sequence $\{\Lambda^{(k)}\}$ is bounded. Suppose $\{\Lambda^{(k_j)}\}$ is a convergent subsequence. Then the corresponding $\{x^{(k_j)}\}$ also has a convergent subsequence. Without causing any ambiguity, we may assume the subsequence $\{(\Lambda^{(k_j)}, x^{(k_j)})\}$ converges. By Theorem 4.3 it follows that the limit point of $\{(\Lambda^{(k_j)}, x^{(k_j)})\}$ is a solution of the MEP. By Theorem 3.8 there are only finitely many such limit points.

For convenience we rewrite the matrix A as a column of m blocks, that is, $A = [A_1, \ldots, A_m]^T$, where $A_i := [A_{i1}, \ldots, A_{im}]$. Then $\lambda_i^{(k)} = \|A_i x^{(k)}\|$. Observe that

$$|\lambda_i^{(k+1)} - \lambda_i^{(k)}| = \big|\, \|A_i x^{(k+1)}\| - \|A_i x^{(k)}\| \,\big| \leq \|A_i(x^{(k+1)} - x^{(k)})\|.$$

From (88) and Theorem 4.2 it follows that $|\lambda_i^{(k+1)} - \lambda_i^{(k)}| \longrightarrow 0$ as $k \longrightarrow \infty$. The assertion now follows by applying Lemma 4.4 to the sequence $\{\lambda_i^{(k)}\}$ for each $i = 1, \ldots, m$. $\square$

THEOREM 4.6. *The sequence $\{x^{(k)}\}$ converges as k goes to infinity.*

Proof. The proof basically is the same as that of Theorem 4.5 since we observe from (88) and Theorem 4.2 that componentwise the difference between $x^{(k+1)}$ and $x^{(k)}$ converges to zero as k goes to infinity. $\square$

We now illustrate the dependence of the method on the starting point. Consider the positive definite matrix

$$A = \begin{bmatrix} 4.3299 & 2.3230 & -1.3711 & -0.0084 & -0.7414 \\ 2.3230 & 3.1181 & 1.0959 & 0.1285 & 0.0727 \\ -1.3711 & 1.0959 & 6.4920 & -1.9883 & -0.1878 \\ -0.0084 & 0.1285 & -1.9883 & 2.4591 & 1.8463 \\ -0.7414 & 0.0727 & -0.1878 & 1.8463 & 5.8875 \end{bmatrix},$$

with $m = 2$, $n_1 = 2$, and $n_2 = 3$. It turns out that the sequence $\{x^{(k)}, \lambda_1^{(k)}, \lambda_2^{(k)}\}$ converges to

$$\begin{aligned} x^* &= [0.9357, 0.3528, -0.9341, 0.3508, 0.0667]^T, \\ \lambda_1^* &= 6.5186, \\ \lambda_2^* &= 8.2116 \end{aligned}$$

if $x^{(0)} = [0.9777, 0.2098, 0.5066, 0.5069, 0.6975]^T$ and to

$$\begin{aligned} x^{**} &= [0.7166, 0.6975, 0.5654, -0.4327, -0.7022]^T, \\ \lambda_1^{**} &= 6.2405, \\ \lambda_2^{**} &= 7.8607 \end{aligned}$$

if $x^{(0)} = [0.7914, 0.6114, 0.4753, 0.2517, -0.8431]^T$. On the other hand, if the method is repeatedly applied to randomly generated starting points, it is interesting that approximately 60% of the points give convergence to x^*, whereas all the remaining converge to x^{**}. This ratio stays about the same regardless of whether a normal distribution or a uniform distribution is used as the generator. This is a clear indication that out of the 24 solutions to this MEP, there are *two* local maxima to the maximal correlation problem. The example also illustrates that Horst's algorithm has a substantial possibility of not converging to the absolute maximal correlation.

5. Conclusion and future research. Although we have demonstrated the convergence of Horst's algorithm for the MEP, it is clear that there are still many other numerical issues worthy of investigation. For example, let

$$\Gamma := \operatorname{diag}\{\gamma_1 I^{[n_1]}, \ldots, \gamma_m I^{[n_m]}\}, \tag{92}$$

where each γ_i is a real number. Then $(A - \Gamma)x = \Lambda x$ if and only if $Ax = (\Gamma + \Lambda)x$. In other words, like the power method, multivariate shifting is a possible strategy for finding other solutions of the MEP. Determining which solution the algorithm converges to on the basis of the starting value for $x^{(0)}$ and the shift parameters is thus an interesting problem. A theoretical problem associated with the multivariate shifting is related to the so-called educational testing problem; that is, for what Γ will the matrix $A - \Gamma$ become positive semidefinite?

On the other hand, the scheme involved in (71) is very analogous to the so-called Jacobi method used in solving linear equations. It is thus rather natural to formulate the following Gauss–Seidel method for the MEP.

ALGORITHM 5.1 (Gauss–Seidel algorithm).

```
Given $x^{(0)} = (x_1^{(0)^T}, \ldots, x_m^{(0)^T})^T$ with $||x_i^{(0)}|| = 1$, do
    for $k = 1, 2, \ldots$
        for $i = 1, \ldots, m$
```

$$y_i^{(k)} := \sum_{j=1}^{i-1} A_{ij} x_j^{(k+1)} + \sum_{j=i}^{m} A_{ij} x_j^{(k)}, \tag{93}$$

$$\lambda_i^{(k)} := ||y_i^{(k)}||, \tag{94}$$

```
(95)        $x_i^{(k+1)} := \frac{y_i^{(k)}}{\lambda_i^{(k)}}.$

        end
    end
```

Let the matrix A be decomposed as

(96) $$A = D + U^T + U,$$

where D is the main diagonal part of A and U is the strictly upper triangular part of A. Then Algorithm 5.1 may be written as

(97) $$(D + U)x^{(k)} = (\Lambda^{(k)} - U^T)x^{(k+1)}.$$

Similarly, an SOR algorithm may be formulated as follows.

ALGORITHM 5.2 (SOR algorithm).

```
Given $x^{(0)} = (x_1^{(0)^T}, \ldots, x_m^{(0)^T})^T$ with $||x_i^{(0)}|| = 1$, do
    for $k = 1, 2, \ldots$
        for $i = 1, \ldots, m$
(98)        $\overline{y}_i^{(k)} := \sum_{j=1}^{i-1} A_{ij}x_j^{(k+1)} + \sum_{j=i}^{m} A_{ij}x_j^{(k)},$
(99)        $\xi_i^{(k)} := ||\overline{y}_i^{(k)}||,$
(100)       $\overline{z}_i^{(k+1)} := \frac{\overline{y}_i^{(k)}}{\xi_i^{(k)}}.$
(101)       $y_i^{(k)} := \omega_i \overline{z}_i^{(k+1)} + (1 - \omega_i)x_i^{(k)},$
(102)       $\lambda_i^{(k)} := ||y_i^{(k)}||,$
(103)       $x_i^{(k+1)} := \frac{y_i^{(k)}}{\lambda_i^{(k)}}.$
        end
    end
```

We note that the relaxation parameters ω_i in each block may be chosen differently. We also remark that the scaling in (100) may be done differently; this includes the possibility of defining $\xi_j^{(k)} := 1$ for all j and all k.

If we define

(104) $$\Xi^{(k)} := \text{diag}\,\{\xi_1^{(k)} I^{[n_1]}, \ldots, \xi_m^{(k)} I^{[n_m]}\}$$

and

(105) $$\Omega := \text{diag}\,\{\omega_1 I^{[n_1]}, \ldots, \omega_m I^{[n_m]}\},$$

then Algorithm 5.2 may be written in matrix form:

$$[(I-\Omega)\Xi^{(k)}+\Omega(D+U)]x^{(k)}=(\Xi^{(k)}\Lambda^{(k)}-\Omega U^T)x^{(k+1)}, \tag{106}$$

which includes (97) as a special case.

Apparently, a partial list of topics that deserves further research should include proof of convergence, rate of convergence, and acceleration of convergence for each of these methods. Work on some of these aspects to the MEP will be expounded on in a forthcoming paper.

REFERENCES

[1] E. L. Allgower and K. Georg, *Simplicial and continuation methods for approximating fixed points and solutions to systems of equations*, SIAM Rev., 22 (1980), pp. 28–85.
[2] S. N. Chow, J. Mallet-Paret, and J. A. Yorke, *Finding zeros of maps: Homotopy methods that are constructive with probability one*, Math. Comput., 32 (1978), pp. 887–899.
[3] M. T. Chu, *A simple application of the homotopy method to symmetric eigenvalue problems*, Linear Algebra Appl., 59 (1984), pp. 85–90.
[4] ———, *Matrix differential equations: A continuous realization process for linear algebra problems*, Nonlinear Anal., 18 (1992), pp. 1125–1146.
[5] ———, *A note on the homotopy method for linear algebraic eigenvalue problems*, Linear Algebra Appl., 105 (1988), pp. 225–236.
[6] M. T. Chu and J. L. Watterson, *Multivariate eigenvalue problems*: II. *Numerical methods*, in preparation.
[7] C. B. Garcia and T. Y. Li, *On the number of solutions to polynomial systems of equations*, SIAM J. Numer. Anal., 17 (1980), pp. 540–546.
[8] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.
[9] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.
[10] H. Hotelling, *The most predictable criterion*, J. Educ. Psychol., 26 (1935), pp. 139–142.
[11] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, 1991.
[12] P. Horst, *Relations among m sets of measures*, Psychometrika, 26 (1961), pp. 129–149.
[13] ———, *Factor Analysis of Data Matrices*, Holt, Rinehart and Winston, New York, 1965.
[14] W. Ledermann, *On the rank of the reduced correlation matrix in multiple factor analysis*, Psychometrika, 2 (1937), pp. 85–93.
[15] T. Y. Li, *On Chow, Mallet-Paret and Yorke homotopy for solving systems of polynomials*, Bulletin Inst. Math. Acad. Sinica, 11 (1983), pp. 433–437.
[16] N. G. Lloyd, *Degree Theory*, Cambridge University Press, New York, 1978.
[17] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1984.
[18] A. P. Morgan, *A homotopy for solving polynomial systems*, Appl. Math. Comput., 18 (1986), pp. 87–92.
[19] P. J. Browne and B. D. Sleeman, *A numerical technique for multiparameter eigenvalue problems*, IMA J. Numer. Anal., 2 (1982), pp. 451–457.
[20] H. J. Wacker, *Continuation Methods*, Academic Press, New York, 1978.
[21] 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.
[22] S. S. Wilks, *Mathematical Statistics*, John Wiley, New York, 1962.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1107–1142, September 1993

HIGH-ORDER AND EFFICIENT METHODS FOR THE VORTICITY FORMULATION OF THE EULER EQUATIONS*

J. S. LOWENGRUB[†], M. J. SHELLEY[‡], AND B. MERRIMAN[§]

Abstract. In this work, the authors develop new methods for the accurate and efficient solution of the two-dimensional, incompressible Euler equations in the vorticity form. Here, the velocity is recovered directly from the Biot–Savart relation with vorticity, and the vorticity is evolved through its transport equation. Using a generalized Poisson summation formula, the full asymptotic error expansion is constructed for the second-order point vortex approximation to the Biot–Savart integral over a rectangular grid. The expansion is in powers of h^2, and its coefficients depend linearly upon only local derivatives of the vorticity. In particular, the second-order term depends only upon the vorticity gradient. Except at second-order, the coefficients also involve rapidly convergent, two-dimensional lattice sums. At second-order, the sum is conditionally convergent, but can be calculated easily and rapidly. Therefore, we can remove the second-order term explicitly from the point vortex approximation to obtain a fourth-order discretization. In the special case of a square grid, the second-order error term is orthogonal to the vorticity gradient. The convective derivative of vorticity is thus calculated to fourth order using the unmodified point vortex approximation and automatically yields a fourth-order evolution. These methods have been implemented. For the vorticity transport equation, the point vortex sums are evaluated very rapidly using the fast Fourier transform (FFT) algorithm, and vorticity gradients are approximated using high-order difference methods. With high resolution, the authors solve numerically the roll-up of a thin layer of vorticity through the Kelvin–Helmholtz instability, and the interaction of two oppositely signed vortices driven together under an external strain flow. In the latter case, a simple time dependence of the grid is introduced to maintain resolution of the flow.

Key words. vorticity formulation, Biot–Savart integral, point vortex discretization

AMS subject classifications. primary 65M25; secondary 76C05

1. Introduction.

In this work, we develop and test new methods for the accurate and efficient solution of the two-dimensional, incompressible Euler equations in the vorticity form. Here, the velocity is recovered directly from the Biot–Savart relation with the vorticity, and the vorticity itself is evolved through its transport equation. This is in itself not a new idea (see [22], for example). However, we perform a new analysis of the point vortex approximation to the Biot–Savart integral, and construct explicitly its asymptotic error expansion for rectangular grids. This analysis is used to develop new methods that are efficient and fourth-order accurate.

Traditional numerical methods for fluid mechanics have concentrated on either the primitive variable formulation of velocity-pressure or the vorticity-stream formulation (see [23], for example). A Poisson equation for either the pressure or the stream function is discretized and inverted at each time step with appropriate boundary conditions suitably accounted for. Advantages of these methods include their rapid evaluation and their overall accuracy. They are not so naturally adaptive as an adaptive grid would require inverting an elliptic equation with nonconstant coefficients.

Lagrangian vortex methods (see [10], as well as other references listed at the end of this paper, for example) also exploit the relationship between velocity and vorticity to

*Received by the editors September, 17, 1991; accepted for publication (in revised form) September 30, 1992. The authors were supported in part by National Science Foundation Mathematical Postdoctoral Fellowships.

[†]Department of Mathematics, Stanford University, Stanford, California 94305. Present address, Department of Mathematics, University of Minnesota, Minneapolis, Minnesota 55455. This work was supported partially by the Office of Naval Research and by the Air Force Office of Scientific Research under grant AFOSR-90-0090.

[‡]Courant Institute of Mathematical Sciences, New York University, New York, New York 10012.

[§]Department of Mathematics, University of California, Los Angeles, California 90024.

pose the Euler equations as a closed system of differential equations for the motion of the curvilinear coordinate system moving with the fluid velocity (flow map). Advantages of the Lagrangian methods include their natural adaptivity, ease of implementation, and the existence of fast inversion algorithms [17]. However, their accuracy is difficult to maintain due to grid distortion and the singular nature of the Biot–Savart kernel.

Ideally, one would like to retain the adaptivity of the Lagrangian methods while controlling the grid motion so as to reduce distortion effects and to retain high-order accuracy. An advantage of the vorticity formulation is that adaptive meshes can be incorporated very easily. An adaptive mesh is included by making a specific choice of an underlying coordinate system. This choice of coordinate system merely introduces metric functions in the evolution equation for the vorticity and a Jacobian term in the Biot–Savart integral. In this way, we avoid numerically inverting an elliptic equation with nonconstant coefficients, as we use a discretization of the exact inversion formula. An important side benefit is that there is no ambiguity about the far-field boundary conditions, for open or periodic problems, since they are built into the formula exactly. We view this work as a partial step in the development of such adaptive methods. However, the special case of rectangular grids is very useful and interesting in its own right.

Using an appropriately generalized Poisson summation formula, we construct the full asymptotic error expansion for the point vortex approximation to the Biot–Savart integral evaluated on a rectangular grid. The existence of the error expansion was proved by Goodman, Hou, and Lowengrub [16] more generally on smooth, curvilinear grids, but the method of proof was not constructive. Thus in the case of a rectangular grid, we have provided an explicit example of the expansion. Beginning at the second order, the expansion is in powers of h^2, the spatial discretization size. The error is local in that the expansion coefficients depend linearly upon derivatives of the vorticity evaluated only at the mesh point at which the velocity is calculated. In particular, the second-order error term depends only upon the gradient of the vorticity. The error also has a global part in that the coefficients involve two-dimensional lattice sums. However, these sums do not depend on the point of evaluation. These sums are also rapidly convergent except, apparently, at the second order where the sum is only conditionally convergent. We show how this lattice sum can be calculated both easily and rapidly. Thus we can remove the second-order term explicitly from the point vortex approximation to obtain a fourth-order discretization. In the special case of a square grid, even this is unnecessary as the second-order error term is orthogonal to the vorticity gradient. The convective derivative of vorticity is thus calculated to fourth order using the unmodified point vortex approximation, and yields automatically a fourth-order evolution. We also find that in the square grid case, the error expansion contains only powers of h^4 after the second-order term. A single Richardson extrapolation to remove the fourth-order term of the point vortex quadrature leaves the second-order and the eighth-order terms remaining. Thus, in the evolution equation, the second-order term falls out to yield eighth-order evolution.

We note that in the case of more general curvilinear grids, extrapolation can be used to generate high-order accurate methods. A difficulty with extrapolation is that although its order of accuracy is higher than the original method, in general, its error coefficients are also larger.

We have also implemented these methods to demonstrate their utility and present numerical results on two difficult test problems. As the point vortex sums over rectangular meshes are discrete convolutions, they are evaluated using the FFT algorithm and the discrete convolution theorem in $O(N^2 \ln N)$ operations where N^2 is the number of mesh points. Vorticity gradients are approximated by high-order difference methods. The first test problem is the nonlinear roll-up of a thin shear layer through the Kelvin–Helmholtz instability. This is a difficult problem as the vorticity develops spatial complexity and steep gradients. The second test problem is the interaction of two oppositely signed vorticies being driven together by an external straining flow. This problem also features strong vorticity gradient production and is relevant to the study of three-dimensional vortex interaction and reconnection [27]. However, the steepest gradients tend to align in a single direction. Consequently, we employ a simple time dependency of the grid to retain resolution of the flow. We compare it to the case where the grid is held fixed. It is a simple, but nontrivial, problem that nicely illustrates how the appropriate choice of time dependency in the grid can greatly improve the calculation.

Our future work includes further consideration of the vorticity transport equation. For example, our time integration method does not make use of the fact that the vorticity equation is a conservation law. The use of methods constructed for such transport equations as essentially nonoscillatory (ENO) methods [19] or other high-order Godunov-type schemes would likely improve the robustness of these methods [7]. It would also be useful to fund the second-order coefficient for the point vortex approximate on a generally smooth, curvilinear coordinate system so that more general fourth-order methods could be constructed without resorting to extrapolation. For the three-dimensional Euler equations, we note that there is a technical difficulty in applying the method developed here to the point vortex approximation. The difficulty involves the sufficiency conditions for applying the special Poisson summation formula for the Biot–Savart kernel (A.29). See Remark 3 at the end of Appendix A. However, we note that the general Poisson summation formula for singular functions (A.9) is still valid. This is currently under study. While the existence of asymptotic error expansions has been proved for the three-dimensional Biot–Savart integral [13], the method is not constructive. Another very important extension is the inclusion of boundaries with and without viscosity. Finally, we note that there is a proof of convergence for a lower-order accurate version of our current method. It will be presented elsewhere.

The outline of the paper is as follows. In §2, we present the method, its implementation, and the special form of its error expansion (the appendices contain the mathematical details of its derivation). In §3, we present some numerical results, and in §4, we give concluding remarks. In Appendix A, the singular Poisson summation formula is derived. In Appendix B, the singular Poisson summation formula is applied to the point vortex discretization of the Biot–Savart integral, and the expansion is proved. In Appendix C, the explicit second-order coefficient is obtained.

2. The vorticity formulation and smooth grid methods.

The Euler equations for two-dimensional, incompressible fluid flow, in the vorticity formulation, are given by

$$\omega_t + \mathbf{u} \cdot \nabla \omega = 0 \tag{2.1a}$$

and

$$\mathbf{u}(\mathbf{x}, t) = \int_{\mathbf{R}^2} \mathbf{K}(\mathbf{x} - \mathbf{x}')\omega(\mathbf{x}', t)dx_1'dx_2', \tag{2.1b}$$

where $\mathbf{u}(\mathbf{x}, t) = (u_1(\mathbf{x}, t), u_2(\mathbf{x}, t))$ is the fluid velocity, $\omega(\mathbf{x}, t)$ is the scalar vorticity, $\mathbf{x} = (x_1, x_2)$, and

$$\mathbf{K}(\mathbf{x}) = \frac{1}{2\pi|\mathbf{x}|^2}(-x_2, x_1) \tag{2.1c}$$

is the Biot–Savart kernel. Equation (2.1a) expresses that the vorticity in two dimensions is conserved along particle paths, while (2.1b) and (2.1c), the Biot–Savart integral, give the relation between the velocity and the vorticity due to the incompressibility constraint and definition of the vorticity. It is assumed that ω is smooth and rapidly decaying at infinity.

Although we will soon restrict ourselves to considering the case of rectangular grids, it is instructive to recast the Euler equations in terms of a more general curvilinear coordinate system. Consider the transformation $\mathbf{X}(\mathbf{b},t) = (x_1(b_1,b_2,t), x_2(b_1,b_2,t))$, which, for example, might arise as the solution to the evolution equation

$$\frac{d}{dt}\mathbf{X}(\mathbf{b},t) = \mathbf{F}(\mathbf{X}(\mathbf{b},t),t) \tag{2.2a}$$

for some $\mathbf{F}$. This yields the formulation

$$\frac{d}{dt}\omega(\mathbf{X}(\mathbf{b},t),t) = [(\mathbf{F}(\mathbf{X}(\mathbf{b},t),t) - \mathbf{u}(\mathbf{X}(\mathbf{b},t),t))] \cdot [[\nabla_{\mathbf{b}}\mathbf{X}]^{-1} \cdot \nabla_{\mathbf{b}}\omega] \tag{2.2b}$$

with

$$\mathbf{u}(\mathbf{X}(\mathbf{b},t),t) = \int_{\mathbf{R}^2} \mathbf{K}(\mathbf{X}(\mathbf{b},t) - \mathbf{X}(\mathbf{b}',t))\omega(\mathbf{X}(\mathbf{b}',t),t)J(\mathbf{b}')db_1'db_2', \tag{2.3}$$

where J is the Jacobian of the coordinate transformation given by

$$J = \det\nabla_{\mathbf{b}}\mathbf{X}.$$

A Lagrangian vortex method [10] uses the specific choice $\mathbf{F} = \mathbf{u}$. In this frame, $\mathbf{X}(\mathbf{b},t)$ is the trajectory of a fluid particle along which the vorticity is conserved. Furthermore, the transformation is area preserving due to incompressibility. An Eulerian vortex method, on the other hand, uses the choice $\mathbf{F} = \mathbf{0}$. Here, $\mathbf{X}(\mathbf{b},t) = \mathbf{b}$ and describes a fixed coordinate system. The vorticity is not conserved in this frame and must be advected. The transformation is trivially area preserving.

There are many possible choices of a quadrature method for this integral. For example, there is the point-vortex approximation [22], [26] or quadrature methods based upon mollification of the singular kernel (see [1], [2], [4], [5], [10], [15], [18], [24], for example). We have chosen a fourth-order quadrature method on a rectangular grid that is related to the point vortex method. It has the advantages of ease of implementation, can be summed by a fast method, and is easily generalized to higher orders.

Let the transformation $\mathbf{X}(\mathbf{b},t)$ be discretized uniformly in $\mathbf{b}$. Define $\mathbf{X}_{\mathbf{j}} = \mathbf{X}(\mathbf{b}_{\mathbf{j}},t)$, with $\mathbf{b}_{\mathbf{j}} = (j_1h, j_2h)$. Similar definitions hold for J and ω. Let $\mathbf{I}[\omega]_{\mathbf{j}}$ denote the value of the Biot–Savart integral evaluated at $\mathbf{X}_{\mathbf{j}}$. The point vortex approximation $\mathbf{I}[\omega]_{\mathbf{j}}^0(h)$ is simply the trapezoidal rule approximation to (2.3), where the singular self-induction term is omitted:

$$\mathbf{I}[\omega]_{\mathbf{j}}^0(h) = h^2 \sum_{k\neq j} \mathbf{K}(\mathbf{X}_{\mathbf{j}} - \mathbf{X}_{\mathbf{k}})J_{\mathbf{k}}\omega_{\mathbf{k}}. \tag{2.4}$$

The error is then

$$\mathbf{E}[\omega]_{\mathbf{j}}^0(h) = \mathbf{I}[\omega]_{\mathbf{j}}^0 - \mathbf{I}[\omega]_{\mathbf{j}}^0(h). \tag{2.5}$$

Even though the integrand of the Biot–Savart integral is singular, it was proved by Goodman, Hou, and Lowengrub [16] that there exists a regular asymptotic expansion in powers of h^2 for the error. That is, for sufficiently small h, there exists constants $\mathbf{C}[\omega]_{\mathbf{j},2p}$ independent of h, such that

$$\mathbf{E}[\omega]_{\mathbf{j}}^{0}(h) = \mathbf{C}[\omega]_{\mathbf{j},2}h^2 + \mathbf{C}[\omega]_{\mathbf{j},4}h^4 + \cdots. \tag{2.6}$$

In general, it appears difficult to find explicit formulae for the coefficients $\mathbf{C}[\omega]_{\mathbf{j},2p}$. For example, in the analysis of Goodman, Hou, and Lowengrub, the second-order coefficient $\mathbf{C}[\omega]_{\mathbf{j},2}$ arises only as an undetermined constant of integration. This particular coefficient is arguably the most useful to know explicitly, as it might then be removed explicitly to yield a higher-order method.

In the special (but very useful) case of a rectangular grid, we give an alternative derivation of the error expansion in which the coefficients $\mathbf{C}[\omega]_{\mathbf{j},2p}$ are given explicitly. We only outline the method here, and refer the interested reader to the appendices for the full details. The main tool is the Poisson summation formula generalized to singular functions. The Poisson summation formula (see, for example, [14]) expresses the trapezoidal sum (say, over the plane) as a sum of Fourier coefficients of the summand. The asymptotic behavior of these Fourier coefficients then allows an analysis of the error in the trapezoidal approximation. However, the integrand of (2.1b) lacks the smoothness necessary for direct application of the Poisson summation formula, which, if used naively, leads to sums of Fourier coefficients that are not absolutely, but only conditionally, convergent. The value of such a sum depends upon how the sum is evaluated. Instead, the integrand of (2.1b) is first smoothed by convolution with an approximate δ-function δ_ϵ of width ϵ. The Poisson summation formula is then applied to the trapezoidal approximation of the smoothed integral and the limit $\epsilon \to 0$ is taken. This leads to the Poisson summation formula generalized to singular functions. The particular choice of δ_ϵ leads to a particular interpretation of the conditionally convergent sum, and produces a compensating residue from the singular self-induction term omitted in (2.4). We have chosen δ_ϵ so as to actually yield rapidly and absolutely convergent sums, and an explicitly calculable residue term.

There is at least an apparent difficulty in applying these methods to the general curvilinear case to find *explicit* expressions for the error coefficients. We would require the Fourier transform of the kernel $\mathbf{K}(\mathbf{X}(\mathbf{b}))$ in the $\mathbf{b}$ variable. This is not generally known.

Let $\mathbf{X}_{\mathbf{j}} = (j_1h_1, j_2h_2) = (j_1\alpha_1h, j_2\alpha_2h)$ be the rectangular grid. Let $\omega_{x_2,\mathbf{j}}$ denote the value of ω_{x_2} evaluated at $\mathbf{X}_{\mathbf{j}}$, and similarly for $\omega_{x_1,\mathbf{j}}$. The point vortex approximation is now simply

$$\mathbf{I}[\omega]_{\mathbf{j}}^{0}(h) = h_1h_2 \sum_{\mathbf{k}\neq\mathbf{j}} \mathbf{K}((j_1-k_1)h_1, (j_2-k_2)h_2)\omega_{\mathbf{k}}. \tag{2.7}$$

We then have the following theorem.

Error Expansion Theorem. *Let $\omega(\mathbf{x})$ be smooth and rapidly decaying and $\alpha = \alpha_1/\alpha_2$.*

Part A. *The second-order error coefficient is given by*

$$\mathbf{C}[\omega]_{\mathbf{j},2}h^2 = \left(\frac{-\omega_{x_2,\mathbf{j}}}{4\pi}C_2(\alpha), \frac{\omega_{x_1,\mathbf{j}}}{4\pi}C_2(1/\alpha)\right) h_1h_2, \tag{2.8a}$$

where

$$C_2(\alpha) = \frac{4}{\pi}\arctan(1/\alpha) - \frac{4\alpha}{\pi}\sum_{m=1}^{\infty}\frac{1}{m}f(m;\alpha), \tag{2.8b}$$

and

$$f(m;\alpha) = \frac{1}{m}\sum_{l=0}^{m}{}' \left(\frac{(l/m)^2-\alpha^2}{((l/m)^2+\alpha^2)^2} + \frac{1-\alpha^2(l/m)^2}{(1+\alpha^2(l/m)^2)^2}\right). \tag{2.8c}$$

By $\sum'$ *is meant the trapezoidal sum; that is, the terms at* $l = 0$ *and* m *are weighted by a factor of* $\frac{1}{2}$.

Remark 1. In the special case of a square grid ($\alpha = 1$), $C_2(\alpha) = C_2(1/\alpha) = 1$, and it follows that

$$\mathbf{C}[\omega]_{\mathbf{j},2} = \frac{1}{4\pi}\nabla^{\perp}\omega|_{\mathbf{j}h}, \tag{2.9}$$

where $\nabla^{\perp}\omega = (-\partial_{x_2}, \partial_{x_1})\omega$ is the perpendicular gradient of ω. Thus, the second-order error term in (2.4) is orthogonal to $\nabla\omega$ (i.e., $h^2\nabla^{\perp}\omega\cdot\nabla\omega = 0$), and the method of lines discretization of (2.1a), is

$$\frac{d}{dt}\omega_{\mathbf{j}}^{h} = -\mathbf{I}[\omega^h]_{\mathbf{j}}^{0}(h)\cdot(\nabla_h\omega^h)_{\mathbf{j}}, \tag{2.10}$$

where ∇_h is an approximation to ∇ to at least fourth order. Equation (2.10) then has a fourth-order consistency error in space, even though the velocity approximation is only second order. That this error is realized in numerical simulations of fluid motions will be demonstrated in the next section.

Remark 2. For the case of the rectangular grid, the calculation of the coefficient $C_2(\alpha)$ appears formidable. And indeed, a straightforward implementation of the sum (2.8b), for $\alpha = 2$, produces summands of the size 10^{-14} only by $m = 12600$. Fortunately, the convergence of this sum can be considerably improved by using the Euler–MacLaurin summation formula (see, for example, [29]) to successively remove the dominant behavior of the summand. From the summation formula applied to (2.8c), we have for $m >> 1$,

$$\begin{aligned} f(m;\alpha) &= \frac{1}{6}\frac{\alpha^2-1}{(1+\alpha^2)^2}\frac{1}{m^2} - \frac{1}{30}\frac{(\alpha^2-1)(\alpha^4-10\alpha^2+1)}{(1+\alpha^2)^4}\frac{1}{m^4} + O\left(\frac{1}{m^6}\right) \\ &= f_2(\alpha)\frac{1}{m^2} + f_4(\alpha)\frac{1}{m^4} + O\left(\frac{1}{m^6}\right). \end{aligned}$$

Thus, we can rewrite the sum in (2.8b) as

$$\begin{aligned} \sum_{m=1}^{\infty}\frac{1}{m}f(m;\alpha) &= \sum_{m=1}^{\infty}\frac{1}{m}\left[f(m;\alpha) - f_2(\alpha)\frac{1}{m^2} - f_4(\alpha)\frac{1}{m^4}\right] \\ &\quad + f_2(\alpha)\zeta(3) + f_4(\alpha)\zeta(5), \end{aligned} \tag{2.11}$$

where

$$\zeta(s) = \sum_{m=1}^{\infty}\frac{1}{m^s}, s > 1,$$

is the Riemann zeta function. In particular, $\zeta(3) = 1.202056903159594$ and $\zeta(5) = 1.036927755143370$. The sum in (2.8b) now converges as $O((1/m)^7)$, and is trivial to compute; for $\alpha = 2$, the above formula produces summands of magnitude 10^{-14} by $m = 45$. Figure 1 shows $C_2(\alpha)$ calculated by the above formulae.

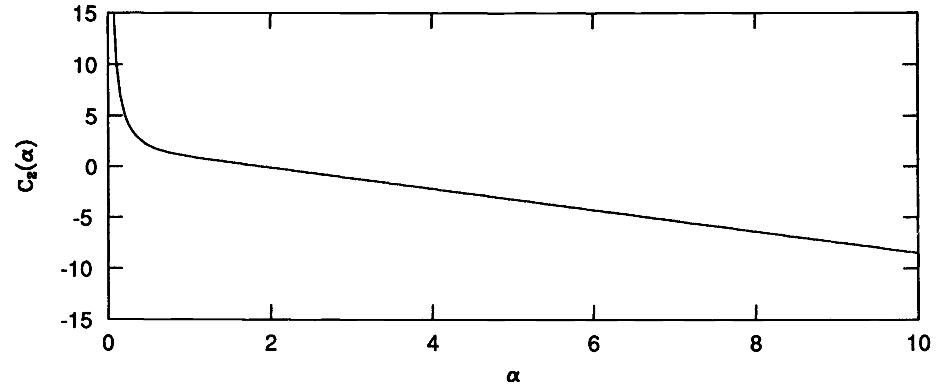


FIG. 1. *The second-order error coefficient* $C_2(\alpha)$.

Remark 3. We can use the calculation of $C_2(\alpha)$ to obtain a fourth-order quadrature rule for the Biot–Savart integral by subtracting the second-order term

$$\tilde{\mathbf{I}}_{\mathbf{j}}(h) = \mathbf{I}[\omega]^0_{\mathbf{j}}(h) - \left(\frac{-\omega^h_{x_2,\mathbf{j}}}{4\pi} C_2(\alpha), \frac{\omega^h_{x_1,\mathbf{j}}}{4\pi} C_2(1/\alpha) \right) h_1 h_2. \tag{2.12}$$

We have assumed that $\omega_{x_2,\mathbf{j}}$ and $\omega_{x_1,\mathbf{j}}$ are approximated to at least fourth order by $\omega^h_{x_2,\mathbf{j}}$ $\omega^h_{x_1,\mathbf{j}}$. Use of this formula requires little effort, since for the calculation of ω_t, these spatial derivatives must be approximated anyway.

Remark 4. That the error coefficients only depend on derivatives of ω at $\mathbf{X_j}$ is true also for the higher-order coefficients. This type of expression for $\mathbf{C}[\omega]_{\mathbf{j},2}$ is also very similar to that derived by Sidi and Israeli [29] for quadrature rules of one-dimensional integrals with Cauchy singularities, and which was subsequently used by Shelley [28] in a study of singularity formation in vortex sheet motion.

Remark 5. The second-order error coefficient (2.8a) can also be found without recourse to the Poisson summation formula. This is done by dividing the quadrature error (2.5) into local and nonlocal contributions. The nonlocal contributions can be estimated by using the Euler–MacLaurin summation formula, and the local contributions lead to sums similar to those in (2.8). It may be that the more general curvilinear case could be handled by such an approach. Furthermore, in the case of the square grid, the expression of $\mathbf{C}[\omega]_{\mathbf{j},2}$, (2.9), can be found by manipulating both integral and sum so as to have a smoother integrand and summand, to which standard quadrature estimates can then be applied.

Part B (*Error Expansion Theorem*). *The higher-order error coefficients* ($p > 1$) *are given by*

$$\mathbf{C}[\omega]_{\mathbf{j},2p} = \frac{1}{(2\pi)^{2p}} \sum_{\substack{r,q \\ r+q=2p-1}} (-D^{q,r}(\alpha)[\omega], D^{r,q}(1/\alpha)[\omega])|_{\mathbf{x_j}}, \tag{2.13}$$

where $D^{r,q}(\alpha)$ is a linear differential operator of order r in x_1 and order q in x_2, and whose coefficients depend upon α, r, and q.

Remark 6. The operator $D^{q,r}(\alpha)$ involves lattice sums analogous to those in (2.8). These sums are absolutely convergent, and their values are independent of the choice of smoothing function δ_ϵ. It is only the second-order coefficient for which there is a question of interpretation of the sum.

Remark 7. In the special case of a square grid ($\alpha = 1$), it can be shown that $\mathbf{C}[\omega]|_{\mathbf{j},2p} = \mathbf{0}$ for $p > 1$ and odd. That is, after the h^2 term, the expansion has only powers of h^4. Thus, an eighth-order method could be easily obtained by performing one Richardson extrapolation of the point vortex approximation to remove the fourth-order term. The second-order and eighth-order terms remain, but the second-order term disappears in the evolution equation as before.

The proof of the error expansion theorem is given in the appendices. In Appendix A the derivation of the Poisson summation formula for singular functions is given. In Appendix B, the extended Poisson summation formula is used to rigorously derive the coefficients $\mathbf{C}[\omega]_{\mathbf{j},2p}$ of the error expansion (2.6). In Appendix C, the second-order error term is expressed in terms of rapidly convergent series as given in (2.8).

It has now been shown how, for rectangular grids, fourth-order methods can be found by a trivial and explicit removal of the second-order error (and even this was not necessary in the square grid case). We also note that the point vortex approximation can be evaluated very efficiently in this case. Equation (2.7) can be written as

$$\mathbf{I}[w]_{\mathbf{j}}^{0}h = \sum_{\mathbf{k}} \tilde{\mathbf{K}}_{\mathbf{j}-\mathbf{k}}\omega_{\mathbf{k}}, \tag{2.14}$$

where

$$\tilde{\mathbf{K}}_{\mathbf{j}} = h_1 h_2 \mathbf{K}(j_1 h_1, j_2 h_2).$$

That is, (2.14) is a discrete convolution. In such cases, through the discrete Fourier transform and its associated discrete convolution theorem, the FFT algorithm can be used to evaluate such sums very rapidly. To employ these techniques, the kernel $\tilde{\mathbf{K}}_{\mathbf{j}}$ and vorticity $\omega_{\mathbf{j}}$ must be extended appropriately as periodic functions, and then represented as discrete Fourier transforms. The details of this are straightforward and will not be given here. For a rectangular mesh the computational complexity of forming the velocity on the mesh is thus $O(N^2 \ln N)$, where N is the number of grid points in each direction. There are now very well developed multidimensional FFT packages that are highly vectorized, and that realize the asymptotic complexity of the algorithm for relatively small values of N.

In the more general case of a curvilinear coordinate system, the existence of the error expansion also allows the derivation of higher-order methods through Richardson extrapolation. Here we give explicitly a fourth-order formula. Using

$$\mathbf{E}[\omega]_{\mathbf{j}}^{0}(2h) = \mathbf{I}[\omega]_{\mathbf{j}} - \mathbf{I}[\omega]_{\mathbf{j}}^{2h} = 4\mathbf{C}[\omega]_{\mathbf{j},2}h^2 + 16\mathbf{C}[\omega]_{\mathbf{j},4}h^4 + \cdots$$

together with (2.5), we find that defining the new quadrature rule $\mathbf{I}[\omega]_{\mathbf{j}}^{1}(h)$ by

$$\mathbf{I}[\omega]_{\mathbf{j}}^{1}(h) = \frac{1}{3}\left[4\mathbf{I}[\omega]_{\mathbf{j}}^{0}(h) - \mathbf{I}[\omega]_{\mathbf{j}}^{0}(2h)\right] = \frac{4}{3}h^2 \sum_{\substack{j_1+k_1 \text{ odd} \\ \text{or} \\ j_2+k_2 \text{ odd}}} \mathbf{K}(\mathbf{X}_{\mathbf{j}} - \mathbf{X}_{\mathbf{k}})J_{\mathbf{k}}\omega_{\mathbf{k}} \tag{2.15}$$

and gives the error

$$\mathbf{E}[\omega]_{\mathbf{j}}^{1}(h) = \mathbf{I}[\omega]_{\mathbf{j}} - \mathbf{I}[\omega]_{\mathbf{j}}^{1}(h) = -12\mathbf{C}[\omega]_{\mathbf{j},4}h^4 + \cdots. \tag{2.16}$$

A method of lines treatment of (2.2b) would then have the form

$$\frac{d}{dt}\omega_{\mathbf{j}}(t) = [(\mathbf{F}(\mathbf{X}_{\mathbf{j}}(t),t) - \mathbf{I}[\omega(t)]_{\mathbf{j}}^{1}(h))] \cdot [\nabla_{h,\mathbf{b}}\mathbf{X}]_{\mathbf{j}}^{-1} \cdot (\nabla_{h,\mathbf{b}}\omega)_{\mathbf{j}}. \tag{2.17}$$

Obviously, further extrapolations to gain higher order could be done computationally, or by calculation of the specific formula. However, we note that further extrapolations would involve computing further doublings of the mesh size, which generally leads to larger error coefficients in the error expansions.

The fourth-order quadrature method given by (2.15), $\mathbf{I}[\omega]_{\mathbf{j}}^{1}(h)$, can be efficiently implemented in a number of ways. If the grid is nonrectangular, then fast summation techniques such as those using hierarchical elements [1], or multipole expansions [17] can be employed. Equation (2.15) can be interpreted as dividing the grid into four interlaced subgrids, as shown in the schematic in Fig. 2. Each grid point is labeled as belonging to subgrid M_1, M_2, M_3, or M_4. Then the quadrature rule $\mathbf{I}[\omega]_{\mathbf{j}}^{1}(h)$ for $\mathbf{b}_{\mathbf{j}} \in M_i$ becomes

$$\mathbf{I}[\omega]_{\mathbf{j}}^{1}(h) = \frac{4}{3}h^2 \sum_{\mathbf{b}_{\mathbf{k}} \notin M_i} \mathbf{K}(\mathbf{X}_{\mathbf{j}} - \mathbf{X}_{\mathbf{k}})J_{\mathbf{k}}\omega_{\mathbf{k}}. \tag{2.18}$$

Thus, this fourth-order quadrature rule for the Biot–Savart integral is seen as an abstracted point vortex approximation, where instead of a singular self-interaction term being omitted, a subgrid of quadrature points is omitted. Also, the set of field points M_1 all use the same set of vortex points M_2, M_3, and M_4 to find the velocity on M_1, and the quadrature weights are independent of the field point position. These are the types of quadrature rules to which fast summation methods are applicable. In principle, fast summation methods based on multipole expansions or hierarchical elements have computational complexities of $O(N^2)$ or $O(N^2 \ln N)$. However, we have not found these methods to be competitive in evaluating the sum when summation using the FFT can be employed.

3. Some numerical results.

We have implemented these methods to numerically solve two incompressible open-flow problems of enduring interest. The first is the nonlinear roll-up of a thin shear layer through the Kelvin–Helmholtz instability. The second is the interaction of two oppositely signed vortices being driven together by an external straining flow.

For a flow periodic in one direction (say, 2π-periodic in x_1), the Biot–Savart integral can be summed exactly over each period to yield an integral, with modified kernel, over a single periodic strip. In particular, $\mathbf{K}$ in (2.1c) is replaced by

$$\mathbf{K}_{\text{per}}(x_1, x_2) = \frac{1}{4\pi}\frac{(-\sinh(x_2), \sin(x_1))}{\cosh(x_2) - \cos(x_1)},$$

and the domain of integration is $[0, 2\pi] \times (-\infty, +\infty)$. We assume that the vorticity is rapidly decaying as $|x_2| \to \infty$.

We consider the initial data

$$\omega(x_1, x_2) = -\exp(-\alpha(x_1)^2 x_2^2),$$

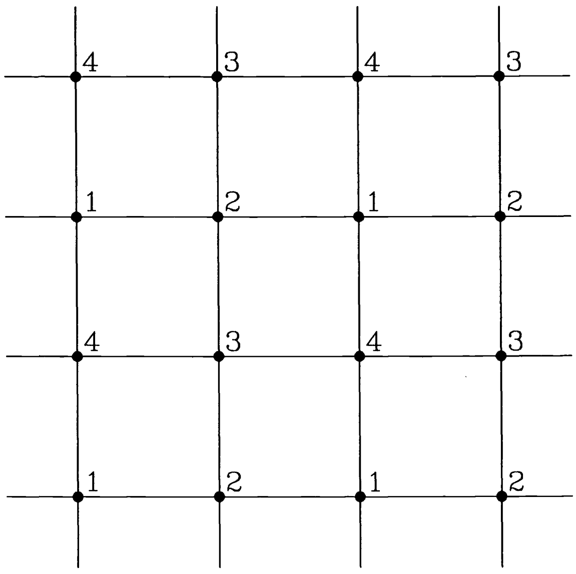


FIG. 2. *A schematic of the four-grid decomposition, M_1, M_2, M_3, and M_4.*

where $\alpha(x_1)^2 = \beta[1+\epsilon(\cos x_1 - 1)]$. That is, ω describes a flat shear layer of characteristic width $1/\sqrt{\beta}$, whose width is sinusoidally perturbed with nondimensional amplitude ϵ. If the vorticity amplitude were likewise scaled as $\sqrt{\beta}$ (fixing the circulation independently of β) and $\beta \to \infty$, this would yield vortex sheet initial data with an initially flat sheet and a sinusoidal perturbation to a constant vortex sheet strength.

We have chosen $\beta = 10$ and $\epsilon = .1$. This is a fairly thin shear layer with aspect ratio of about 20:1. As this flow develops very steep gradients in the vorticity, it is a severe test of any method. Figures 3(a)–3(d) show contours of the evolving vorticity at the times $t = 0$, 20, 25, and 30, respectively. The computational domain is $[0, 2\pi] \times [-5\pi/8, 5\pi/8]$ and vorticity is initially of the size $2 \cdot 10^{-17}$ on the upper and lower boundaries. Figures 3(a)–3(c) are with $N_1 = 512$ and $N_2 = 320$, and Fig. 3(d) is with $N_1 = 1024$ and $N_2 = 640$. The tick marks on the box boundaries correspond to the grid. These choices for N_1 and N_2 give a square mesh covering the computational domain. The contour lines are from -0.95 up to -0.05 at intervals of 0.05.

Here, the velocity on the grid is computed by the point vortex approximation (2.7) using the periodic kernel $\mathbf{K}_{\text{per}}$, and the sums are evaluated rapidly using the FFT. By scaling both N_1 and N_2 as N, the amount of work per time step is then $O(N^2 \ln N)$. On a Cray Y-MP, with $N_1 = 1024$ and $N_2 = 640$, evaluation of the velocity took about one second. The vorticity gradient is computed by cubic splines in each direction, assuming periodic boundary conditions in the x_1-direction, and zero data on the upper and lower boundaries. Cubic splines with such boundary conditions yield fourth-order derivative approximations on the mesh points. Then, since $h_1 = h_2$, and by Remark 1 of §2, the method of lines discretization (2.10) of the vorticity transport equation (2.1a) has a fourth-order spatial consistency error. The method of time integration used is a fourth-order, Adams–Moulton predictor-corrector method. The time step was $\Delta t = .00625$, which, for the $N_1 = 1024$ calculation, was halved at $t = 20$ to satisfy the CFL condition.

The time steps used are small enough to effectively remove time discretization errors from the calculation.

Figures 3(a)–3(d) trace the usual course of vortex layer roll-up (see, for example, [3] for such calculations with layers of constant vorticity). The Kelvin–Helmholtz instability initially causes vorticity to advect towards the center, leading to a relative concentration there (see Fig. 3(b)). The layer rolls up subsequently and forms a central rotating vortex with trailing arms. As the vorticity is strained into the center, the outer arms thin and large gradients in the vorticity are created there. As the contour lines are also material curves for the flow, their considerable stretching and folding likewise lead to large gradients.

The error in the calculations can be estimated through use of its asymptotic expansion. For example, by assuming that for the vorticity

$$E^h_{\omega,\mathbf{j}} = \omega_{\mathbf{j}} - \omega^h_{\mathbf{j}} = A_4h^4 + A_6h^6 + \cdots, \tag{3.1}$$

we then have the estimate

$$E^h_{\omega,\mathbf{j}} = \frac{\omega^{2h}_{\mathbf{j}} - \omega^h_{2\mathbf{j}}}{2^4 - 1} + O(h^6). \tag{3.2}$$

Then

$$L_{h,\infty}[\omega] = \max_{\mathbf{j}} \left| \frac{\omega^{2h}_{\mathbf{j}} - \omega^h_{2\mathbf{j}}}{2^4 - 1} \right| \tag{3.3}$$

is an approximation to maximum error in the vorticity.

The left-hand box in Fig. 4 shows $L_{h,\infty}[\omega]$ (solid) and $L_{h,\infty}[u]$ (dashed) for $h = 2\pi/N_1$ with $N_1 = 128, 256, 512$, and 1024 for $0 \le t \le 30$, while the right-hand box shows the order of the calculations approximated by the expression $\log_2(L_{2h,\infty}/L_{h,\infty})$. Here u is the velocity in the x_1-direction and is calculated on the grid using (2.12). All the evidence suggests a stable and convergent scheme. For a single calculation, the error shows a monotonic increase as the layer rolls up. This is hardly surprising (referring to Figs. 3(a)–3(d)); gradients of vorticity evolving under the Euler equations grow very rapidly, and in fact obey an evolution equation similar to that for the vorticity vector in three dimensions. As the resolution is improved, the accuracy is improved upon the entire range of integration. The approximation of the order shows a clear fourth-order error for a range of time that increases as the resolution is improved. The velocity error shows generally a slower decrease than does the vorticity error. This is true as well for the decrease away from fourth-order error. This is likely a consequence of the velocity being one degree smoother than the vorticity.

A difficulty of the $L_{h,\infty}$ measure of the error in ω^h is that it assumes that ω^{2h} is still well represented by (3.1). Other measures of the error, which do not depend as strongly on expansion (3.1), are the approximated circulation and maximum vorticity. For this flow the circulation C in a periodic strip is a conserved quantity. The graph at the left in Fig. 5 shows $-\log_{10}|C - C_h|$, as a function of time, for the four successive doublings of the mesh size beyond $N_1 = 64$. C_h is calculated using the trapezoidal rule over the computational box, which would be of infinite order in the absence of discretization errors in ω (this explains the very high accuracy at $t = 0$). The calculated order of the convergence is shown in the graph at right, and the fourth-order convergence is seen clearly over nearly the entire range of integration. We note however that the circulation is an integral measure of the error, and as such is a weak measure of accuracy.

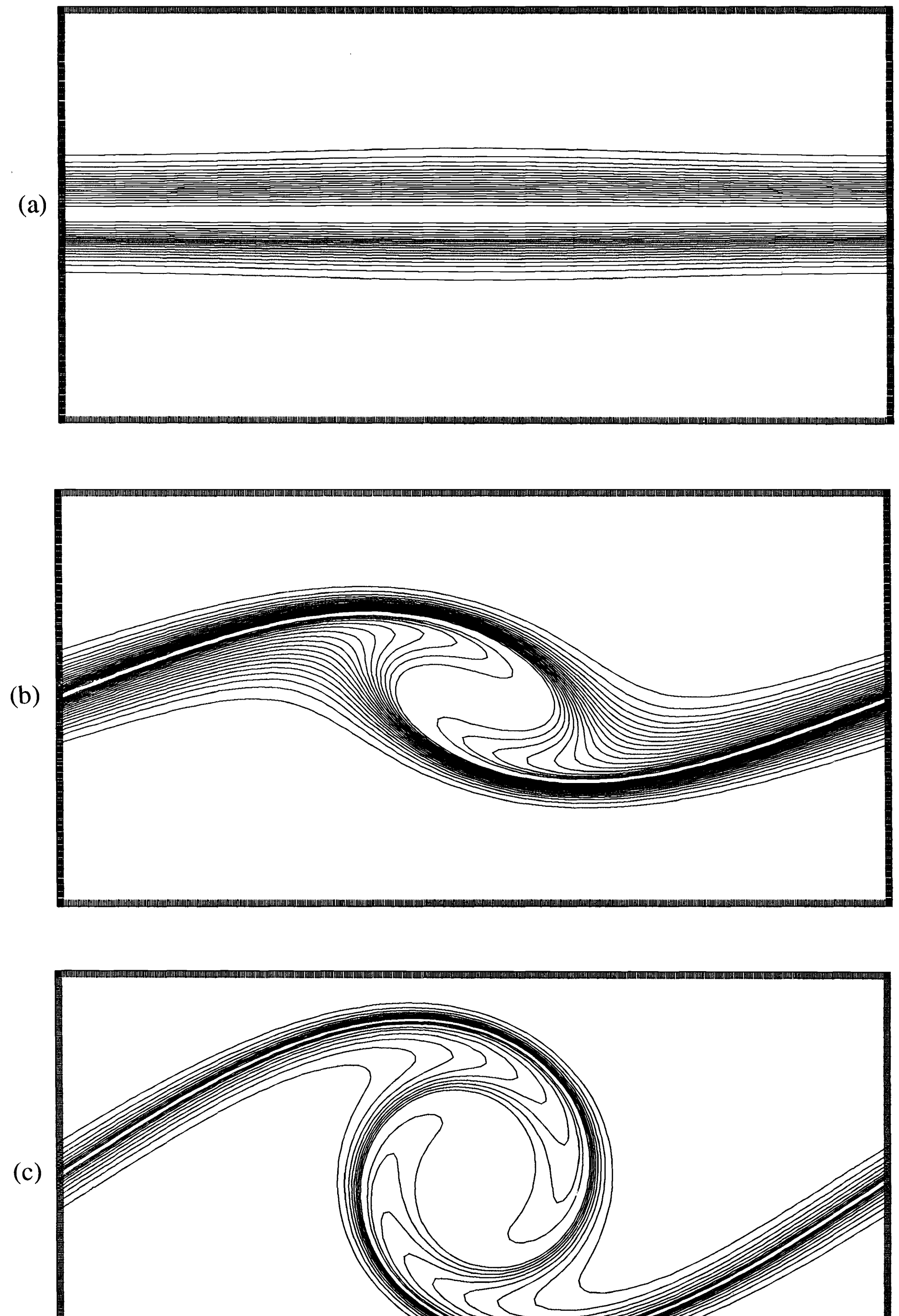


FIG. 3. *Contours of the vorticity for an evolving thin shear layer at times $t = 0, 20, 25,$ and 30. The contour levels run from -0.95 to $-.05$, at intervals of .05.*

(d)

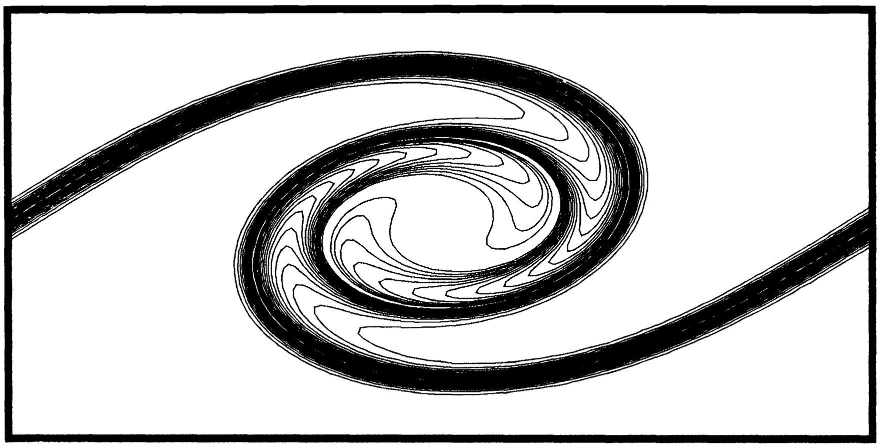


FIG. 3. *Continued.*

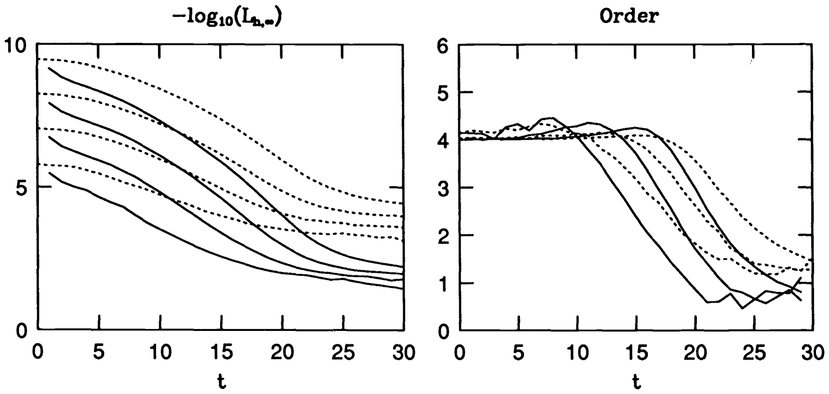


FIG. 4. *The approximated error in vorticity (solid) and velocity (dashed) for $h = 2\pi/N_x$ with $N_x = 128, 256, 512,$ and 1024 for $0 \le t \le 30$. The accuracy improves as N is increased. The right-hand box shows the order of the calculations.*

Another invariant of the two-dimensional Euler equations is $\|\omega\|_\infty$, the maximum modulus of the vorticity. This quantity must be estimated point-wise. As such, it is a stronger measure of error than the circulation. Figure 6 shows $\|\omega\|_{h,\infty}$, or the maximum vorticity on the mesh, for the five different resolutions. For the initial condition used, $\|\omega\|_\infty = 1$, and $|\omega(x_1, x_2, t = 0)| = 1$ along the entire curve $x_2 = 0$. While convergence is plainly seen, the spatial location of the maximum value jumps around the mesh as the flow evolves, and it is difficult to estimate an order of convergence from this quantity. We note specifically, that for the calculation with $N_1 = 1024$ and $N_2 = 640$, $\|\omega\|_{h,\infty} \simeq 1.001$ at $t = 30$.

We have also performed these calculations using the more general fourth-order point vortex approximation (2.15), obtained by extrapolation of the second-order quadrature. The extrapolated method also shows fourth-order convergence, and appears stable

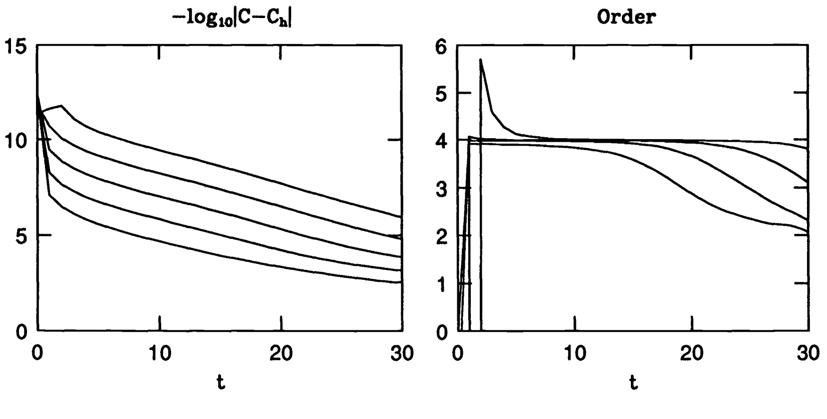


FIG. 5. *The error in circulation, with its approximation calculated by using the trapezoidal rule over the computational box for* $N_x = 64, 128, 256, 512$, *and* 1024. *The accuracy improves as N is increased. The right-hand box shows the orders of the computations.*

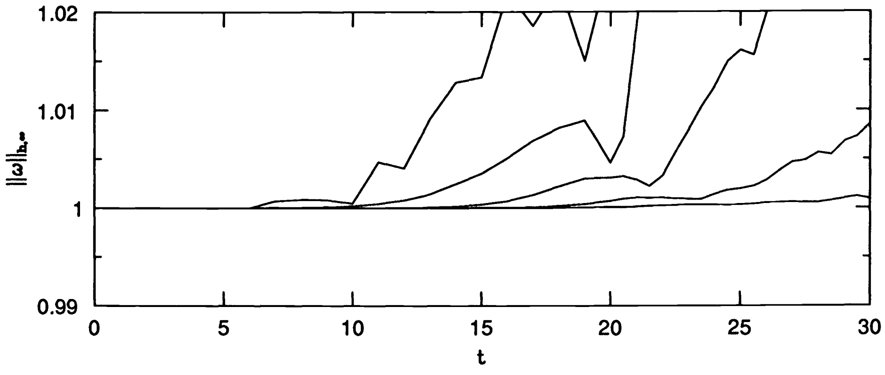


FIG. 6. *The maximum vorticity on the grid* $\|\omega\|_{h,\infty}$ *for* $N_2 = 64, 128, 256, 512$, *and* 1024. *The difference from* 1 *decreases as N is increased.*

and convergent. Its associated errors are typically larger by about a factor of twelve than those associated with using (2.10), which is consistent with the change in the fourth-order error coefficient introduced by the extrapolation.

We now consider a much different problem. We calculate the motion of two oppositely signed vortices that interact under the imposition of an external straining flow. This problem features a strong vorticity gradient production that quickly taxes any method. However, the steepest gradients tend to be aligned in a single direction, and the introduction of a simple time-dependency of the grid helps retain resolution of the flow. The behavior of such flows is relevant to the study of three-dimensional vortex interaction and reconnection (see, for example, [27]). It is a simple, but nontrivial, problem that nicely illustrates how appropriate choice of time dependency in the grid motion can greatly improve the accuracy of a calculation.

Consider special solutions to the three-dimensional Euler equations that are of the form

$$
\begin{aligned}
\mathbf{U} &= (u, v, w) \\
&= (\tilde{u}(x_1, x_2, t) - \gamma x_1, \tilde{v}(x_1, x_2, t) + \beta x_2, \tilde{\omega}(x_1, x_2, t) + (\gamma - \beta)x_3).
\end{aligned} \tag{3.4}
$$

Here, while there are components of velocity and vorticity in all three directions, the only true x_3-dependency is through the irrotational and incompressible straining field

$$(-\gamma x_1, \beta x_2, (\gamma - \beta)x_3).$$

For solutions of this form, the full Euler equations reduce to two-dimensional evolution equations for u and v, in which the only effect of the velocity component w is through the rate-of-strain in that direction, namely, $\gamma - \beta$. Here, w is simply advected as a scalar field in the two-dimensional velocity field $\mathbf{V} = (\tilde{u} - \gamma x_1, \tilde{v} + \beta x_2)$. In terms of the scalar vorticity $\omega = v_{x_1} - u_{x_2} = \tilde{v}_{x_1} - \tilde{u}_{x_2}$, the evolution equations can be given in a form very similar to equation set (2.1). Namely,

$$\omega_t + \mathbf{V} \cdot \nabla\omega = \omega(\gamma - \beta), \tag{3.5a}$$

and

$$(\tilde{u}, \tilde{v})(\mathbf{x}, t) = \int_{\mathbf{R}^2} \mathbf{K}(\mathbf{x} - \mathbf{x}')\omega(\mathbf{x}', t)dx_1'\, dx_2', \tag{3.5b}$$

where $\mathbf{K}$ is defined as before by (2.1c). As initial data we take

$$\omega(\mathbf{x}, 0) = a(\exp((x_1 - 1/2)^2 + x_2^2)/d^2 - \exp((x_1 + 1/2)^2 + x_2^2)/d^2). \tag{3.6}$$

The initial vorticity is then two Gaussian cores of opposite sign and with width d, centered at $\pm 1/2$. The rates-of-strain are set as $\gamma = 2$ and $\beta = 1$. That is, there is a compressive external flow in the x_1-direction that pushes the two vortices together and extensional flows in the x_2 and x_3-directions. The extensional rate-of-strain in the x_2-direction elongates the vortices in that direction, while that in x_3 leads to vortex stretching. Furthermore, we choose $a = 10$ and $d = 1/4$.

Such a flow could be considered as a model for the interaction of two adjacent vortices being driven together by strains induced by other vorticity distributions in the flow, or as a model of the effect of three-dimensional vortex tube curvature, where the two vortices are cross sections of two adjacent vortex tubes. Such a flow illustrates the effect of external strains producing core deformation and thus altering the interaction of the vortices with each other.

The primary difficulty in computing such a flow is that as the two vortices are driven together by the external straining flow, large gradients are rapidly produced in the vorticity. To maintain good resolution of the flow, we found it necessary to introduce a simple time dependence of the grid. Namely, we allowed the grid to collapse with the compressive strain in the x_1-direction, so as to maintain resolution of the sharp vertical gradients that have formed. In addition, as two vortices translate vertically through their mutually induced velocities, the x_2-position of the grid must be chosen accordingly so that the vorticities are kept within the computational domain. That is,

$$\frac{d}{dt}x_1(\mathbf{b}, t) = -\gamma x_1(\mathbf{b}, t) \quad \text{with } x_1(\mathbf{b}, t = 0) = b_1,$$

$$\frac{d}{dt}x_2(\mathbf{b}, t) = \bar{v}_2(t) \quad \text{with } x_2(\mathbf{b}, t = 0) = b_2,$$

where $\bar{v}_2(t)$ is chosen to keep the x_2-centroid of the vortices fixed at $x_2 = 0$. This requires the computation of the circulation, the x_2-centroid and its time derivative all on a half-plane of the domain (since the circulation over the whole domain is zero). We calculated these integrals on the right half-plane by Simpson's rule. This choice maintains a rectangular grid.

For this time-dependent coordinate system we can find $\nabla_b \mathbf{X}$ in closed form, and thus also the Jacobian J for (2.3). Thus, we can evolve the vorticity transport equation (2.2b) (with the forcing term from the strain included). More importantly, the grid is no longer square, and the point vortex approximation (2.7) does not yield a fourth-order evolution without modification. Accordingly, we now use the quadrature rule (2.12), where $C_2(\alpha(t)), \alpha = h_1/h_2$, must be calculated at each time step. As noted in Remark 2 of §2, this can be done easily and rapidly through the use of formula (2.11), and only requires $\nabla\omega$, whose computation is already necessary. Again, it is assumed that ω is zero on the boundaries. For variety we use fourth-order differences rather than splines to calculate $\nabla\omega$. This appears to have little effect on the computation.

Figures 7(a)–7(c) show the contours of the computed vorticity field at $t = 0, .5$ and 1.0 with $N_1 = N_2 = 256$ and $\Delta t = .00078125$. The perimeter in Figs. 7(a)–7(c) is the boundary of the time-dependent computational domain. The computational box is initially of length 4.2 on a side. The time step was chosen small enough to effectively remove time discretization errors from this calculation (as well as the calculation where the initial x_1-positions of the grid were held fixed).

Figure 8 shows the errors in ω (solid) and u (dashed), as well as their orders, as approximated by (3.3). The errors in ω are normalized by the maximum vorticity, $\|\omega\|_\infty(t) = a\exp(\gamma - \beta)t$. Here, $N_1 = N_2 = 128$ and 256. This is to be contrasted with Fig. 9, which shows the same quantities, but for the same calculations with the initial x_1-grid locations held fixed. The errors for the calculation with the moving grid is much smoother in time, and increase more slowly. Indeed, by $t = 0.4$, the moving grid calculation with $N_1 = 128$ is as accurate as that for the fixed grid with $N_1 = 256$, and becomes more accurate (by virtue of its slower increase) by the end of the calculation.

The total conserved circulation of the two vortices is zero. However, it is also true that circulation computed from vorticity of a single sign is conserved, and in particular the circulation in the left or right half-plane is conserved (note that $x_1 = 0$ is a free-slip surface for this flow). Figure 10 shows the error in the circulation for the right-hand vortex (calculated on the right half-plane using Simpson's rule), and its computed order, for the moving (solid) grid calculation at the three resolutions $N_1 = N_2 = 64, 128$, and 256 and the fixed (dashed) grid at the $N_1 = N_2 = 256$ resolution. This figure only serves to further illustrate the advantage in accuracy the moving grid affords over the fixed calculation. There is a slight degradation in accuracy near $t = 1$ for the $N_1 = N_2 = 256$ calculation. This is due to the vertical boundary (upon which the vorticity and its derivatives are assumed to be zero) coming too close to the support of the vorticity. The lower resolution runs are less sensitive to such a boundary effect. Figure 11 shows the normalized maximum vorticity on the mesh

$$\|\omega\|_{h,\infty}/\|\omega\|_\infty = ae^{-(\gamma-\beta)t}\|\omega\|_{h,\infty}$$

for the moving grid calculations (solid). The fixed grid quantity with $N_1 = N_2 = 256$ is included as the dashed curve. Again, convergence is observed as N is increased. The normalized maximum vorticity for the $N_1 = N_2 = 256$ calculation with moving grid is 0.9985 at $t = 1$.

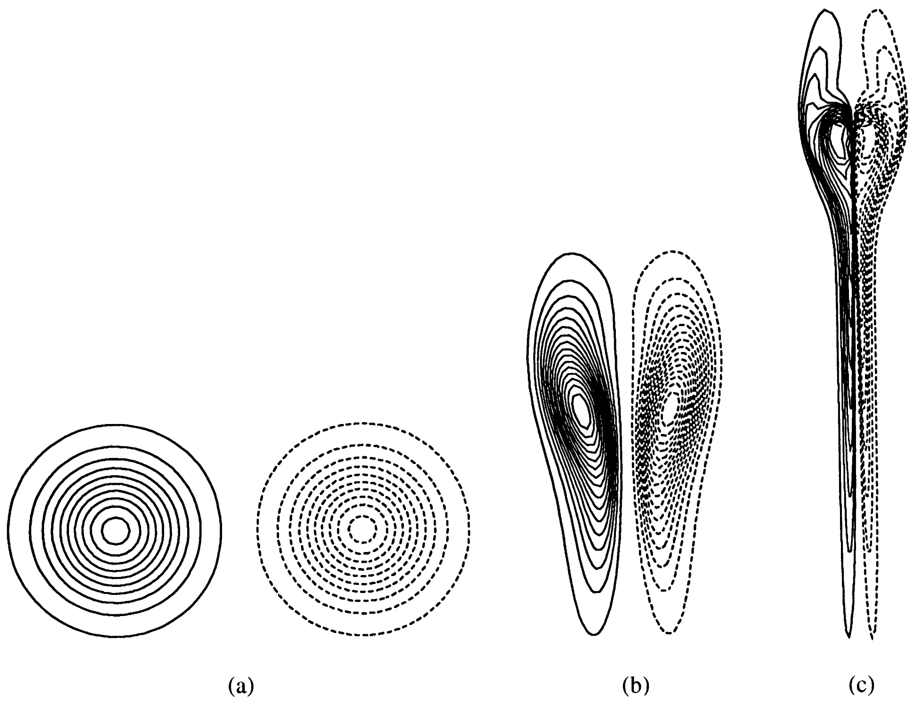


FIG. 7. *Contours of the vorticity for the two-vortex straining flow at* $t = 0, .5,$ *and* 1. *The contour levels in Fig.* 7(a) *from* -9.4 *to* 9.6, *in Fig.* 7(b) *from* -16.3 *to* 16.4, *and in Fig.* 7(c) *from* -28.4 *to* 28.5, *all with increments of* 1. *Here, the initial box size is* 4.2 *on a side,* $N_x = N_y = 256$, *and* $\Delta t = .00078125$.

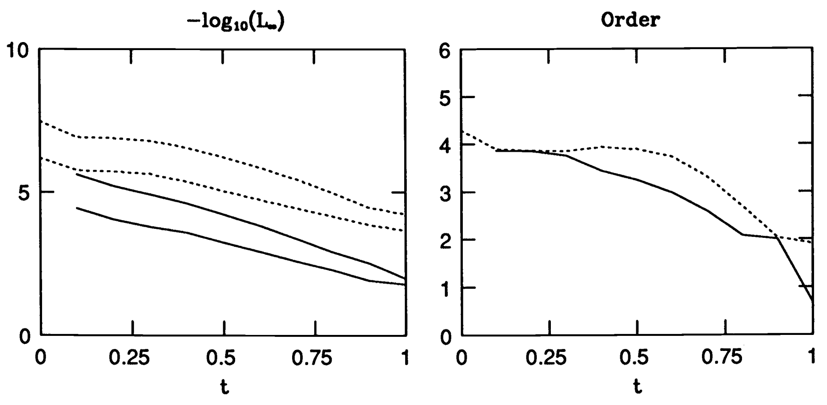


FIG. 8. *The approximated error in vorticity (solid) and velocity (dashed) with their orders in the right-hand box for* $N_x = N_y = 128, 256$. *The errors in vorticity are normalized by the maximum vorticity* $\|\omega\|_\infty(t) = a\exp(\gamma - \beta)t$. *The accuracy improves as N is increased. The right-hand box shows the orders of the computations.*

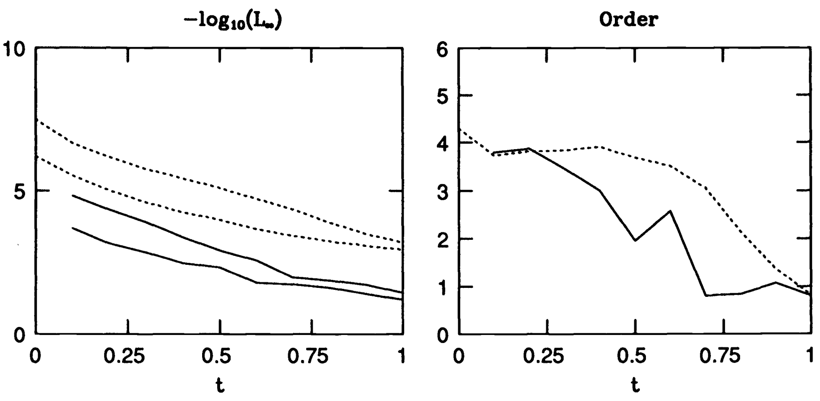


FIG. 9. *The same quantities as in Fig. 7, but with the initial grid held fixed.*

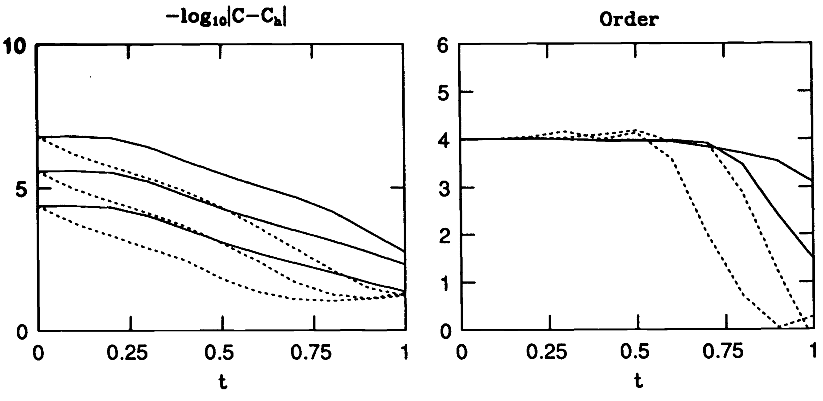


FIG. 10. *The circulation error in the right half-plane, with its approximation computed by using Simpson's rule in the right half-plane. The* solid *curves are for the* moving *grid and the* dashed *curves are for the* fixed *grid. The errors are for $N_x = N_y = 64, 128, 256$. The accuracy improves as N is increased. The order of these errors is presented in the right-hand box.*

Finally, we note that in recent work, Fishelov [15] also considered a fixed-grid method for the vorticity formulation of the two-dimensional, incompressible Euler equations. She used a vortex blob discretization evaluated by direct summation. This approach may be more useful for long times and for cases where the grid is allowed to move since the blob discretization provides improved stability for Lagrangian methods (see [5], for example). However, the blob approach is not likely to be useful for fixed grids and for short times, as we found that a vortex blob discretization with a Beale and Majda fourth-order kernel (using $\delta = h^{.95}$) gave velocity errors that were two orders of magnitude larger than those for our fourth-order modified point vortex discretization (2.12) at $t = 0$ and for the initial data (3.6) and $N_1 = N_2 = 256$.

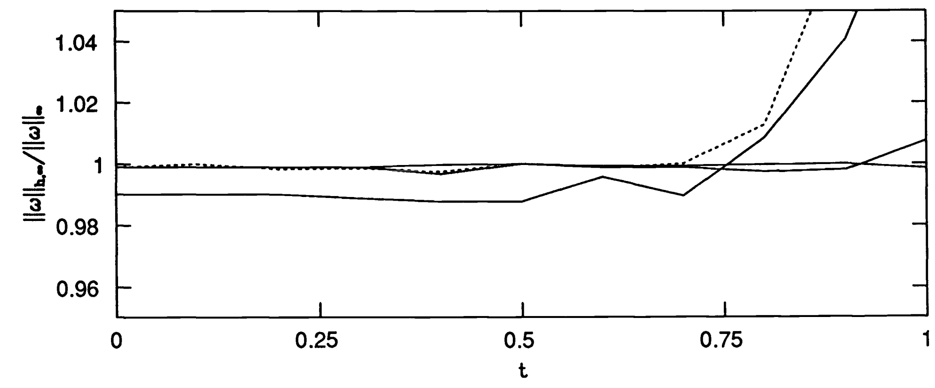


FIG. 11. *The normalized maximum vorticity on the grid,* $\|\omega\|_{h,\infty}/\|\omega\| = ae^{-(\gamma-\beta)t}\|\omega\|_{h,\infty}$. *The* solid *curves are for the* moving *grid with* $N_x = N_y = 64, 128, 256$, *and the* dashed *curve is for the* fixed *grid with* $N_x = N_y = 256$. *The difference from* 1 *decreases as N is increased.*

4. Conclusion. In this work we have developed new numerical methods, based on the vorticity formulation, for solving the two-dimensional, incompressible Euler equations. In the vorticity formulation of the Euler equations, the velocity is obtained directly from the vorticity through the Biot–Savart integral, and the vorticity is updated on the computational grid. An advantage of this formulation for open and periodic problems is that asymptotic boundary conditions on the velocity are explicitly incorporated into the Biot–Savart kernel, and the focus is then simply on finding accurate and efficient methods for evaluating the Biot–Savart integral over a regular mesh. Here we have considered the point vortex approximation on rectangular meshes.

Using the Poisson summation formula generalized to include singular integrands, we constructed the explicit error expansion for the point vortex approximation to the Biot–Savart integral. The existence of the error expansion was proved by Goodman et al. more generally on smooth, curvilinear grids, but the method of proof was not constructive. Thus in the case of a rectangular grid, we have proved an explicit example of the expansion. We found that the leading second-order error term depends on the vorticity only through the local value of its gradient. The coefficient is expressible as a two-dimensional lattice sum, and we demonstrate how its calculation can be done easily and rapidly. This leads to a simple modification of the point vortex approximation that yields fourth-order accuracy. In the special case of a square grid, the method of lines formulation of the vorticity transport equation, using the unmodified, second-order point vortex approximation and a fourth-order gradient approximation, automatically yields a fourth-order evolution of the vorticity. We note also that in the square grid case, the error expansion contains terms with only fourth powers of h after the leading second-order term.

Furthermore, we have demonstrated the utility of these methods through their employment to solve both the nonlinear roll-up of a thin, spatially periodic shear layer, and the interaction of two oppositely signed vortices being pushed together under an external, three-dimensional straining flow. As the summations were done on a rectangular grid, the FFT algorithm was used (through use of the discrete convolution theorem) to rapidly evaluate the velocity on the grid. Thus, the calculations are performed at high resolution. We were able to accurately follow the evolution of the shear layer well into the nonlinear regime, where the bulk of the vorticity has concentrated itself into a central vortex. This concentration is linked, in the thin layer limit, to the formulation of singularities observed in vortex sheets [3]. By employing a grid that collapsed naturally with the external compressive flow, we were able to compute the motion of two vortices driven together under the strain. Physically, the interest lies in their severe deformation and elongation, which yields enhanced dissipation in the presence of viscosity (see, for example, [9] for viscous calculations).

These methods have already found application in the study of buoyancy-driven flows, including both two-dimensional Boussinesq and fully stratified fluids [25]. In both these cases, the incompressibility constraint gives the Biot–Savart relation between velocity and vorticity, and the methods are directly applicable. There are several other directions that further work should take. Our time integration method does not make use of the fact that the vorticity transport equation is a conservation law. The use of methods constructed for such transport equations, such as ENO methods [19], or other high-order Godunov type schemes, would improve the robustness of these methods (see also [7]). It would also be useful to find the second-order error coefficient for the point vortex approximation on a generally smooth, curvilinear coordinate system. This would allow the construction of fourth-order methods together with more general adaptivity, without resorting to explicit extrapolation to gain additional accuracy. For the three-dimensional Euler equations, we have seen that there is an apparent difficulty in directly applying the methods developed here to the so-called grid-free point vortex approximation. We are currently studying this. We note that the existence of asymptotic error expansions has been proved not only for the three-dimensional Biot–Savart integral, but also for its gradient whose computation is necessary to compute the stretching term in the vorticity transport equation (see [13]). Another very important extension is to devise methods to treat the presence of walls, both with and without the presence of viscosity.

Appendix A. Poisson summation formula for singular functions. In this appendix, we present a version of the usual Poisson summation formula (see [14], for example) extended to the case of a function with an isolated and integrable singularity at the origin and in two space dimensions. It will be clear from this analysis how to obtain more general results. We begin by introducing some notation.

Define the Fourier transform in two dimensions by

$$\hat{f}(\mathbf{k}) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-i\mathbf{k}\cdot\mathbf{x}} f(\mathbf{x})\,dx_1 dx_2, \tag{A.1}$$

where $\mathbf{k} = (k_1, k_2)$ and $\mathbf{x} = (x_1, x_2)$. Its inverse transform is given by

$$f(\mathbf{x}) = \frac{1}{(2\pi)^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-i\mathbf{k}\cdot\mathbf{x}} \hat{f}(\mathbf{x})\,dk_1 dk_2. \tag{A.2}$$

Define the convolution of the functions f, g by

$$f * g(\mathbf{x}) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(\mathbf{x}-\mathbf{x}')g(\mathbf{x}')\,dx_1' dx_2'. \tag{A.3}$$

By the convolution theorem,

$$\widehat{f * g}(\mathbf{k}) = \hat{f}(\mathbf{k})\hat{g}(\mathbf{k}) \tag{A.4}$$

and

$$\widehat{fg}(\mathbf{k}) = \frac{1}{(2\pi)^2}\hat{f} * \hat{g}(\mathbf{k}). \tag{A.5}$$

Denote a smoothing function by $\delta_\epsilon(\mathbf{x})$, and let it satisfy the usual properties of an approximate identity [31]. Then for any f, a smoothed version of it is given by

$$f_\epsilon(\mathbf{x}) = f * \delta_\epsilon(\mathbf{x}). \tag{A.6}$$

Define the physical space grid by

(A.7)

$$\Lambda(\alpha_1,\alpha_2) = \{\mathbf{x_j} \,|\mathbf{x_j} = (j_1h_1, j_2h_2) \text{ with } h_1 = \alpha_1 h, h_2 = \alpha_2 h \quad\text{and}\quad \mathbf{j} = (j_1,j_2) \in \mathbf{Z}^2\},$$

and the dual phase space grid by

$$\Lambda^*(\alpha_1,\alpha_2) = \left\{\mathbf{k_j} \mid \mathbf{k_j} = \left(\frac{1}{\alpha_1}j_1, \frac{1}{\alpha_2}j_2\right) \quad\text{and}\quad \mathbf{j} = (j_1,j_2) \in \mathbf{Z}^2\right\}. \tag{A.8}$$

Then we obtain the following formula.

POISSON SUMMATION FORMULA FOR SINGULAR FUNCTIONS. *Let $f(\mathbf{x})$ be rapidly decaying in* $\mathbf{x}$, *and smooth except for an isolated and integrable singularity at* $\mathbf{x} = \mathbf{0}$. *Then,*

$$\begin{aligned}\sum_{\mathbf{x}\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} f(\mathbf{x_j})h_1h_2 - \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(\mathbf{x})dx_1dx_2 = \\ \lim_{\epsilon\to 0}\left[\sum_{\mathbf{k_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}_\epsilon\left(\frac{2\pi}{h}\mathbf{k_j}\right) - f_\epsilon(\mathbf{0})h_1h_2\right],\end{aligned} \tag{A.9}$$

where f_ϵ is the smoothed version of f.

Remark 1. As the function f has a singularity at the origin, its Fourier coefficients may decay too slowly to be absolutely summable. Consequently, the smoothing function $\delta_\epsilon(\mathbf{x})$ provides the mechanism to interpret their sum. In addition, the smoothing also gives rise to a residue, or self-induction term. However, since the left-hand side of (A.9) is independent of the choice of smoothing function, so is the right-hand side. Thus all smoothing effects in it must cancel.

Remark 2. Under certain circumstances, depending on the function f *and* the smoothing function δ_ϵ, the limit of the right-hand side of (A.9) exists for each term separately. We discuss this in detail after the proof.

Remark 3. Equation (A.9) can be generalized to functions with nonintegrable singularities provided they can be interpreted in the principle value sense. In this case, the choice of smoothing functions is restricted to the ones that give the correct principle value in the continuous case, i.e., in the integral on the left-hand side of (A.9) is replaced by

$$\text{P.V.}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(\mathbf{x})dx_1dx_2,$$

while the rest of the formula (A.9) remains unchanged.

Remark 4. The formula (A.9) can also be generalized to functions with several isolated singularities. For example, suppose that the singular points are $\mathbf{z}_1,\ldots,\mathbf{z}_n$. Let the smoothed version of f be f_ϵ as before. If the singular points lie on the grid $\Lambda(\alpha_1,\alpha_2)$, then (A.9) becomes

$$\sum_{\substack{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)\\ \mathbf{x_j}\neq z_1,\ldots,z_n}} f(\mathbf{x_j})h_1h_2 - \int\int f(\mathbf{x})dx_1dx_2 = \lim_{\epsilon\to 0}\left[\sum_{\mathbf{k_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}_\epsilon\left(\frac{2\pi}{h}\mathbf{k_j}\right) - \sum_{l=1}^{n} f_\epsilon(\mathbf{z}_l)h_1h_2\right]. \tag{A.10}$$

Proof of summation formula. The proof of (A.9) is almost trivial. We rely on the Poisson summation formula for smooth functions [12] as stated in the following.

POISSON SUMMATION FOR SMOOTH FUNCTIONS. *Let $g(\mathbf{x})$ be smooth and rapidly decaying. Then,*

$$\sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)} g(\mathbf{x_j})h_1h_2 - \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} g(\mathbf{x})dx_1dx_2 = \sum_{\mathbf{k_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{g}\left(\frac{2\pi}{h}\mathbf{k_j}\right). \tag{A.11}$$

Now we apply this formula to f_ϵ, which is the smoothed version of f. Thus, we have

$$\sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)} f_\epsilon(\mathbf{x_j})h_1h_2 - \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f_\epsilon(\mathbf{x})dx_1dx_2 = \sum_{\mathbf{k_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}_\epsilon\left(\frac{2\pi}{h}\mathbf{k_j}\right). \tag{A.12}$$

Now write

$$\sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)} f_\epsilon(\mathbf{x_j})h_1h_2 = \sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} f_\epsilon(\mathbf{x_j})h_1h_2 + f_\epsilon(\mathbf{0})h_1h_2. \tag{A.13}$$

Therefore, we get

$$\sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} f_\epsilon(\mathbf{x_j})h_1h_2 - \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f_\epsilon(\mathbf{x})dx_1dx_2 = \sum_{\mathbf{k_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}_\epsilon\left(\frac{2\pi}{h}\mathbf{k_j}\right) - f_\epsilon(\mathbf{0})h_1h_2. \tag{A.14}$$

The limit as $\epsilon\to 0$ of the left-hand side of (A.14) clearly exists. Consequently, the limit as $\epsilon\to 0$ of the right-hand side exists and yields the result (A.9).

It is natural here to address the question of whether the limit of the right-hand side of (A.9) can be taken for each term separately. It is sufficient to determine whether the limit as $\epsilon\to 0$ of $f_\epsilon(\mathbf{0})$ exists. If it does, the existence of the limit of the left-hand side of (A.9) implies that the limit of the sum of Fourier coefficients must then also exist and hence the limit can be separated.

Suppose that $f(\mathbf{x})$ can be written as

$$f(\mathbf{x}) = H(\mathbf{x})g(\mathbf{x}), \tag{A.15}$$

where $H(\mathbf{x})$ contains the local singularity and $g(\mathbf{x})$ is smooth and rapidly decaying. Then we have

$$f_\epsilon(\mathbf{0}) \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} H(\mathbf{x})g(\mathbf{x})\delta_\epsilon(\mathbf{x})dx_1dx_2. \tag{A.16}$$

Define a characteristic function χ_R by

$$\chi_R(\mathbf{x}) = \begin{cases} 1 \text{ if } |x_1| \leq R \text{ and } |x_2| \leq R, \\ 0 \text{ otherwise}. \end{cases} \tag{A.17}$$

Furthermore, define $\tilde{g}(\mathbf{x})$ by

$$\tilde{g}(\mathbf{x}) = g(\mathbf{x}) - [g(\mathbf{0}) + \nabla g(\mathbf{0}) \cdot \mathbf{x}]\chi_R(\mathbf{x}). \tag{A.18}$$

Then $f_\epsilon(\mathbf{0})$ becomes

$$\begin{aligned} f_\epsilon &= g(\mathbf{0}) \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} H(\mathbf{x})\delta_\epsilon(\mathbf{x})\chi_R(\mathbf{x}) dx_1 dx_2 \\ &\quad + \nabla g(\mathbf{0}) \cdot \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \mathbf{x} H(\mathbf{x})\delta_\epsilon(\mathbf{x})\chi_R(\mathbf{x}) dx_1 dx_2 \\ &\quad + \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} H(\mathbf{x})\tilde{g}(\mathbf{x})\delta_\epsilon(\mathbf{x}) dx_1 dx_2 \\ &\equiv I_{\epsilon,R} + II_{\epsilon,R} + III_{\epsilon,R}. \end{aligned} \tag{A.19}$$

Since $H(\mathbf{x}), \mathbf{x}H(\mathbf{x})$, and $H(\mathbf{x})\tilde{g}(\mathbf{x})$ have successively smoother behavior as $|\mathbf{x}| \to 0$, there is no hope for cancellation of divergent terms by summing the different integrals. Therefore, the limit as $\epsilon \to 0$ of $f_\epsilon(\mathbf{0})$ exists if and only if the limit as $\epsilon \to 0$ of $I_{\epsilon,R}$, $II_{\epsilon,R}$, and $III_{\epsilon,R}$ exists separately.

Consider $I_{\epsilon,R}$ first.

$$\lim_{\epsilon \to 0} I_{\epsilon,R} = \lim_{\epsilon \to 0} \int_{-R}^{+R} \int_{-R}^{+R} H(\mathbf{x})\delta_\epsilon(\mathbf{x}) dx_1 dx_2. \tag{A.20}$$

Since $H(\mathbf{x})$ has a local singularity at the origin, the only way such a limit can exist is by some cancellation. Therefore, suppose that

$$H(\mathbf{x}) = -H(-\mathbf{x}), \qquad \delta_\epsilon(\mathbf{x}) = \delta_\epsilon(-\mathbf{x}). \tag{A.21}$$

Then, $I_{\epsilon,R} = 0$ for any ϵ and R and hence the limit (A.20) exists and is equal to 0.

Consider $III_{\epsilon,R}$ next. By construction, $H(\mathbf{x})\tilde{g}(\mathbf{x}) = O(|\mathbf{x}|)$ as $|\mathbf{x}| \to 0$ and is integrable. Therefore,

$$\lim_{\epsilon \to 0} III_{\epsilon,R} = \lim_{\epsilon \to 0} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} H(\mathbf{x})\tilde{g}(\mathbf{x})\delta_\epsilon(\mathbf{x}) dx_1 dx_2 = 0 \tag{A.22}$$

by standard arguments such as the Riemann–Lebesgue lemma if δ_ϵ is the Dirichlet kernel (see below).

Finally, it remains to consider $II_{\epsilon,R}$. The existence of this limit is a classical question in harmonic analysis. In general, this limit will exist if and only if $\mathbf{x} = \mathbf{0}$ is a Lebesgue or a Dirichlet point (see [30]) of the function $\mathbf{x}H(\mathbf{x})$. We do not give the most generally known conditions for the existence of such points here, but it is easy to show that if $H(\mathbf{x}) = K_i(\mathbf{x})$ then these limits exist. K_1 with $i = 1, 2$ denotes the components of the Biot–Savart kernel.

For simplicity, we consider two standard types of smoothing functions: scaled and dilated L^1 functions

$$\delta_\epsilon(\mathbf{x}) = \frac{1}{\epsilon^2}\eta\left(\frac{x_1}{\epsilon}, \frac{x_2}{\epsilon}\right) \tag{A.23}$$

with η even, positive, and of mean 1, and the Dirichlet kernel

$$\delta_\epsilon(\mathbf{x}) = \delta_{\frac{1}{M}}(\mathbf{x}) = \frac{\sin\left(\frac{Mx_1}{\alpha_1}\right)}{\pi x_1}\frac{\sin\left(\frac{Mx_2}{\alpha_2}\right)}{\pi x_2}, \tag{A.24}$$

where α_1, α_2 serve to weight the coordinate directions (see Appendix C).

Now correspondingly, write $II_{\epsilon,R}$ as $II^s_{\epsilon,R}$ for the scaled smoothing function

$$II^s_{\epsilon,R} = \int_{-\frac{R}{\epsilon}}^{+\frac{R}{\epsilon}}\int_{-\frac{R}{\epsilon}}^{+\frac{R}{\epsilon}} \tilde{\mathbf{H}}(\epsilon\mathbf{x})\eta(\mathbf{x})dx_1dx_2, \tag{A.25}$$

where $\tilde{\mathbf{H}}(\mathbf{x}) = \mathbf{x}H(\mathbf{x})$, and as $II^D_{\epsilon,R}$ for the Dirichlet kernel

$$II^D_{\frac{1}{M},R} = \frac{1}{\pi^2}\int_{-\frac{RM}{\alpha_2}}^{+\frac{RM}{\alpha_2}}\int_{-\frac{RM}{\alpha_1}}^{+\frac{RM}{\alpha_1}} \tilde{\mathbf{H}}\left(\frac{\alpha_1 x_1}{M}, \frac{\alpha_2 x_2}{M}\right)\frac{\sin(x_1)}{x_1}\frac{\sin(x_2)}{x_2}dx_1dx_2. \tag{A.26}$$

The question reduces to whether the limit as $\epsilon \to 0$ exists for $II^s_{\epsilon,R}$ or $II^D_{\epsilon,R}$. Now take $H(\mathbf{x}) = \mathbf{K}_i(\mathbf{x})$. Then notice that with this choice, $\tilde{\mathbf{H}}(\mathbf{x})$ is homogeneous of degree 0 so that

$$II^s_{\epsilon,R} = \int_{-\frac{R}{\epsilon}}^{+\frac{R}{\epsilon}}\int_{-\frac{R}{\epsilon}}^{+\frac{R}{\epsilon}} \tilde{\mathbf{H}}(\mathbf{x})\eta(\mathbf{x})dx_1dx_2 \tag{A.27}$$

and

$$II^D_{\frac{1}{M},R} = \frac{1}{\pi^2}\int_{-\frac{RM}{\alpha_2}}^{+\frac{RM}{\alpha_2}}\int_{-\frac{RM}{\alpha_1}}^{+\frac{RM}{\alpha_1}} \tilde{\mathbf{H}}(\alpha_1 x_1, \alpha_2 x_2)\frac{\sin(x_1)}{x_1}\frac{\sin(x_2)}{x_2}dx_1dx_2. \tag{A.28}$$

It is clear that the limit as $\epsilon \to 0$ of $II^s_{\epsilon,R}$ exists since $\tilde{\mathbf{H}}(\mathbf{x})$ is bounded on the plane and $\eta \in L^1(\mathbf{R}^2)$. The limit as $M \to \infty$ of $II^D_{\frac{1}{M},R}$ is directly calculable by contour integration (see Appendix C). Consequently, the limit of the right-hand side of (A.9) can be separated as follows.

SPECIAL POISSON SUMMATION FORMULA FOR THE BIOT–SAVART KERNEL. *If $f(\mathbf{x}) = K_i(\mathbf{x})g(\mathbf{x})$, g is smooth and rapidly decaying, and δ_ϵ is either the scaled smoothing function or the Dirichlet kernel, then*

$$\begin{aligned}\sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} f(\mathbf{x_j})h_1h_2 - \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(\mathbf{x})dx_1dx_2 &= \\ \text{P.V.}\sum_{\mathbf{k_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}\left(\frac{2\pi}{h}\mathbf{k_j}\right) - \lim_{\epsilon\to 0} f_\epsilon(\mathbf{0})h_1h_2&,\end{aligned} \tag{A.29}$$

where the P.V. *denotes the principal value sum defined by*

$$\text{P.V.}\sum_{\mathbf{k}_j\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}\left(\frac{2\pi}{h}\mathbf{k_j}\right) = \lim_{\epsilon\to 0}\sum_{\mathbf{k}_j\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{f}\left(\frac{2\pi}{h}\mathbf{k_j}\right). \tag{A.30}$$

Remark 1. We use this formula (A.29) with $g(\mathbf{x}) = \omega(\mathbf{x})$ in Appendices B and C to construct the explicit error expansion.

Remark 2. Although the result (A.29) is independent of the smoothing function, the rate of convergence as $\epsilon \to 0$ is highly dependent on the choice. In Appendix C, we exploit this property by using the Dirichlet kernel as the smoothing function. This yields rapidly and absolutely convergent sums and thus enables us to obtain very accurate approximations of error expansion coefficients.

Remark 3. Of course, the formula (A.29) holds for more general functions $f = Hg$. Generally, if the limit as $\epsilon \to 0$ of $I_{\epsilon,R}$, II_ϵ, R, and $III_{\epsilon,R}$ exists, then (A.29) holds, although even this is not the most general case. Notice, though, that the three-dimensional version of $II_{\epsilon,R}$ does not have a limit since there the Biot–Savart kernel is $O(1/|x|^2)$ as $|x| \to 0$ so that the corresponding $\mathbf{x}H(\mathbf{x})$ is unbounded at the origin. This is currently under study.

Appendix B. Error Expansion. In this appendix we obtain an error expansion for the point vortex approximation to the Biot–Savart integral on a rectangular mesh. The expansion is obtained through an asymptotic analysis of the Fourier coefficients appearing in the special Poisson formula for the Biot–Savart kernel (A.29).

In previous work, Goodman, Hou, and Lowengrub [16] proved the existence of such an error expansion. However, it is difficult to determine the coefficients of the expansion from their method of proof. For example, their second-order coefficient contains an undetermined constant of integration. Our proof is similar to theirs in spirit, but our method provides a prescription for the explicit determination of the coefficients.

In this appendix, we do not derive explicit expressions for the coefficients, although we indicate how it is done. We save their explicit evaluation for Appendix C where we obtain an expression for the second-order coefficient. This term is the most delicate to obtain. The Biot–Savart kernel in two dimensions is

$$\text{(B.1)} \qquad \mathbf{K}(\mathbf{x}) = \frac{1}{2\pi}\frac{(-x_2, x_1)}{{x_1}^2 + {x_2}^2},$$

where $\mathbf{x} = (x_1, x_2)$. We want to describe the discretization error between the Biot–Savart integral and the point vortex approximation

$$\text{(B.2)} \qquad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{K}(\mathbf{x_l} - \mathbf{x})\omega(\mathbf{x})dx_1dx_2 - \sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{x_l}} \mathbf{K}(\mathbf{x_l} - \mathbf{x_j})\omega(\mathbf{x_j})h_1h_2,$$

where $\Lambda(\alpha_1, \alpha_2)$ is the rectangular grid defined in (A.7), $h_1 = \alpha_1 h$ and $h_2 = \alpha_2 h$.

It would be routine to describe this error if the integrand were smooth. However, $\mathbf{K}(\mathbf{x})$ has an isolated integrable singularity at $\mathbf{x} = \mathbf{0}$. Thus, to handle this, we use the special Poisson summation formula (A.29).

Without loss of generality, and for the sake of clarity, we study the error $\mathbf{I}(\mathbf{0}) - \mathbf{I}_h(\mathbf{0})$, where

$$\text{(B.3)} \qquad \mathbf{I}(\mathbf{0}) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{K}(\mathbf{x})\omega(\mathbf{x})dx_1dx_2$$

and

$$\text{(B.4)} \qquad \mathbf{I}_h(\mathbf{0}) = \sum_{\mathbf{x_j}\in\Lambda(\alpha_1,\alpha_2)\backslash\mathbf{0}} \mathbf{K}(\mathbf{x_j})\omega(\mathbf{x_j})h_1h_2.$$

This corresponds to taking $\mathbf{x}_1 = \mathbf{0}$ in (B.2) and dropping the minus sign inside the kernel. Now, following the prescription of Appendix A, w define the smoothed version of the integrand in (B.3) by

$$(\mathrm{B.5}) \qquad (\mathbf{K}\omega)_\epsilon(\mathbf{x}) = (\mathbf{K}\omega) * \delta_\epsilon(\mathbf{x}) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{K}(\mathbf{x}')\omega(\mathbf{x}')\delta_\epsilon(\mathbf{x}-\mathbf{x}')dx_1'dx_2',$$

where $\delta_\epsilon(\mathbf{x})$ is given by (A.23) or (A.24).

The purpose of this section is now to prove the following result.

THEOREM 1 POINT VORTEX ERROR EXPANSION. *For $\omega(\mathbf{x})$ smooth and rapidly decaying, there exist coefficients $\tilde{\mathbf{C}}(\alpha)[\omega]_{0,2n}$ such that*

$$(\mathrm{B.6}) \qquad \mathbf{I}(\mathbf{0}) - \mathbf{I}_h(\mathbf{0}) = \lim_{\epsilon\to 0}(\mathbf{K}\omega)\,\epsilon(\mathbf{0})h_1h_2 + \sum_{n=1}^{N-1}\left(\frac{h}{2\pi}\right)^{2n}\tilde{C}(\alpha)[\omega]_{0,2N} + O(h^{2N})$$

for any N, α is the grid aspect ratio α_1/α_2. Furthermore, the coefficients $\tilde{C}$ are given by

$$(\mathrm{B.7}) \qquad \tilde{\mathbf{C}}(\alpha)[\omega]_{0,2n} = \sum_{\substack{r,q\\ r+q=2n-1}}\left(-D^{r,q}(\alpha)[\omega](\mathbf{0}), D^{q,r}\left(\frac{1}{\alpha}\right)[\omega](\mathbf{0})\right),$$

and $D^{r,q}(\alpha)$ is a linear differential operator of order r in x_1 and q in x_2 and whose coefficients depend on α, r, q.

Remark 1. For $n > 1$, these coefficients are the coefficients shown in (2.6) and (2.13) of §2. The second-order term $\mathbf{C}(\alpha)[\omega]_{0,2}$ in (2.8a) differs from $\tilde{\mathbf{C}}(\alpha)[\omega]_{0,2}$ exactly by the self-induction term in (B.6).

Remark 2. Our method of proof relies on the application of the special Poisson summation formula for the Biot–Savart kernel (A.29), asymptotics of the resulting Fourier lattice sums, and controlling the error from the asymptotics. The method is very general and can be applied in many other contexts where singular integrals arise.

Proof. We begin with the following lemma.

DISCRETIZATION LEMMA. *Suppose that $\omega(\mathbf{x})$ is smooth and rapidly decaying. Then the discretization error $\mathbf{I}(\mathbf{0}) - \mathbf{I}_h(\mathbf{0})$ is given by*

$$(\mathrm{B.8}) \qquad \mathbf{I}(\mathbf{0}) - \mathbf{I}_h(\mathbf{0}) = \lim_{\epsilon\to 0}(\mathbf{K}\omega)_\epsilon(\mathbf{0})h_1h_2 - \mathrm{P.V.}\sum_{\mathbf{l_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}}\widehat{\mathbf{K}\omega}\left(\frac{2\pi}{h}\mathbf{l_j}\right),$$

where

$$(\mathrm{B.9}) \quad \mathrm{P.V.}\sum_{\mathbf{l_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}}\widehat{\mathbf{K}\omega}\left(\frac{2\pi}{h}\mathbf{l_j}\right) = \lim_{\epsilon\to 0}\sum_{\mathbf{l_j}=\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}}\widehat{\mathbf{K}\omega}\left(\frac{2\pi}{h}\mathbf{l_j}\right)\hat{\delta}_\epsilon\left(\frac{2\pi}{h}\mathbf{l_j}\right).$$

The proof of this lemma is a straightforward application of (A.29) with $\mathbf{f}_\epsilon = (\mathbf{K}\omega)_\epsilon$.

Thus, to determine the error expansion, we must now examine the lattice sum that appears on the right-hand side of (B.8).

Define the function

$$(\mathrm{B.10}) \qquad \mathbf{F}(l_1, l_2, m_1, m_2, \tau) = (F_1, F_2) = \frac{(-l_2 + m_2\tau, l_1 - m_1\tau)}{(l_1 - m_1\tau)^2 + (l_2 - m_2\tau)^2}$$

with $\tau = \frac{h}{2\pi}$ as a small parameter. We now use $\mathbf{F}$ to evaluate $\widehat{\mathbf{K}\omega}(\frac{2\pi}{h}\mathbf{l})$ as follows.

LEMMA. *The Fourier transform of* $\mathbf{K}\omega$ *is*

$$\widehat{\mathbf{K}\omega}\left(\frac{2\pi}{h}\mathbf{l}\right) = \frac{-i}{(2\pi)^2}\tau \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{F}(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1, dm_2, \tag{B.11}$$

where $\tau = \frac{h}{2\pi}$.

Proof. Direct calculation gives

$$\hat{\mathbf{K}}(\mathbf{l}) = -i\frac{(-l_2, l_1)}{l_1^2 + l_2^2} \tag{B.12}$$

with $\mathbf{l} = (l_1, l_2)$. Now, by the convolution theorem, we get

$$\widehat{\mathbf{K}\omega}(\mathbf{l}) = \frac{1}{(2\pi)^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \hat{\mathbf{K}}(\mathbf{l} - \mathbf{m})\hat{\omega}(\mathbf{m})dm_1 dm_2. \tag{B.13}$$

Then, combining (B.12), (B.13), and (B.10) gives the result (B.11).

Now, using the result (B.11) in (A.9), we get

$$\begin{aligned}\text{P.V.} \sum_{\mathbf{l_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \widehat{\mathbf{K}\omega}\left(\frac{2\pi}{h}\mathbf{l_j}\right) &= -\tau \lim_{\epsilon\to 0} \sum_{\mathbf{l_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{\delta}_\epsilon\left(\frac{\mathbf{l_j}}{\tau}\right)\frac{i}{(2\pi)^2} \\ &\quad\cdot \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{F}(l_{j,1}, l_{j,2}, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1 dm_2,\end{aligned} \tag{B.14}$$

where $\mathbf{l_j} = (l_{j,1}, l_{j,2})$. Thus, we need to consider the integral

$$\frac{i}{(2\pi)^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{F}(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1, dm_2, \tag{B.15}$$

where $\mathbf{l} \in \Lambda^*(\alpha_1, \alpha_2)\backslash\mathbf{0}$. As τ is a small parameter, we want to expand (B.15) in τ (recall that $\tau = h/2\pi$). It is this expansion that, combined with the self-induction term, yields the error expansion. Of course, the function $\mathbf{F}$ has a singularity, and so we must be careful how we do this.

FOURIER LATTICE SUM EXPANSION LEMMA. *Suppose that* $\omega(\mathbf{x})$ *is smooth and rapidly decaying, and define* α *to be the aspect ratio of the grid* $\alpha = \alpha_1/\alpha_2$. *Then we obtain the following expansion for the Fourier lattice sum* (B.14):

$$\text{P.V.} \sum_{\mathbf{l_j}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \widehat{\mathbf{K}\omega}\left(\frac{2\pi}{h}\mathbf{l_j}\right) = \sum_{n=1}^{N-1}\left(\frac{h}{2\pi}\right)^{2n} \tilde{\mathbf{C}}(\alpha)[\omega]_{0,2n} + O(h^{2n}) \tag{B.16}$$

for any N, *where*

$$\tilde{\mathbf{C}}(\alpha)[\omega]_{0,2n} = \sum_{\substack{r,q \\ r+q=2n-1}} \left(-D^{r,q}(\alpha)[\omega](\mathbf{0}), D^{q,r}\left(\frac{1}{\alpha}\right)[\omega](\mathbf{0})\right), \tag{B.17}$$

and $D^{r,q}(\alpha)$ *is a linear differential operator of order* r *in* x_1 *and* q *in* x_2 *whose coefficients depend on* α, r, q.

Remark 1. As we will see, the differential operators $D^{r,q}$ arise because an expansion of $\mathbf{F}$ in (B.15) leads to the calculation of moments of $\hat{\omega}$. These moments are then identified with the derivatives of ω.

Remark 2. Although the coefficients $\tilde{C}(\alpha)[\omega]_{0,2n}$ depend on principle value lattice sums, only the value of the second-order term ($n = 1$) will change according to the smoothing function. This term involves a lattice sum that is not absolutely summable and thus its value will depend on how it is summed. For $n > 1$, there is no ambiguity in how the sum is taken as the relevant lattice sums are absolutely convergent.

Remark 3. As a result of the special sensitivity of the second-order term, we will discuss it in more detail in Appendix C. In particular, we will see how its value depends on the type of smoothing. Of course, as the resulting expansion is independent of the smoothing function, the self-induction term compensates for the dependence on the smoothing function here. This is analogous to the undetermined constant of integration in the analysis of Goodman, Hou, and Lowengrub [16].

Remark 4. The symmetry of the coefficients $\tilde{C}$ in α and r, q is a result of the relationship between F_1 and F_2, i.e., $F_2(l_1, l_2, m_1, m_2, \tau) = -F_1(l_2, l_1, m_2, m_1, \tau)$.

Remark 5. In the square grid case $\alpha = 1$, it can be shown that $\tilde{C}(\alpha)[\omega]_{0,2n} = \mathbf{0}$ for n odd. Thus, but for the second-order term, expansion is in powers of h^4. We do not present this here.

Proof. We begin by examining $\mathbf{F}$. Notice that $\mathbf{F}$ has a singular point at

$$(m_1, m_2) = \left(\frac{l_1}{\tau}, \frac{l_2}{\tau}\right). \tag{B.18}$$

Thus, since τ is a small parameter, we can expand $\mathbf{F}$ away from the singular point (B.18) in a series about $\tau = 0$:

$$\mathbf{F}(l_1, l_2, m_1, m_2, \tau) = \sum_{n=0}^{\infty} \partial_\tau^n \mathbf{F}(l_1, l_2, m_1, m_2, 0)\frac{\tau^n}{n!}. \tag{B.19}$$

We define a box in m-space about the singular point (B.18) by

$$\mathbf{B} = \left[\frac{l_1 - \tau}{\tau}, \frac{l_1 + \tau}{\tau}\right] \times \left[\frac{l_2 - \tau}{\tau}, \frac{l_2 + \tau}{\tau}\right]. \tag{B.20}$$

Then, consider the integral (B.15):

$$\begin{aligned}
&\frac{i}{(2\pi)^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathbf{F}(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1 dm_2 \\
&= \frac{i}{(2\pi)^2}\int\int_{\mathbf{R}^2\setminus B} \mathbf{F}(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1 dm_2 \\
&+\frac{i}{(2\pi)^2}\int\int_{B} \mathbf{F}(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1 dm_2 \\
&\equiv I + II.
\end{aligned} \tag{B.21}$$

Now, the expansion (B.19) of $\mathbf{F}$ can be used in I. We will show later that II is of arbitrarily high order in $\tau = (h/2\pi)$. Using the expansion, we write I as

$$I = \sum_{n=0}^{\infty} \frac{\tau^n}{n!}\frac{i}{(2\pi)^2}\int\int_{\mathbf{R}^2\setminus B} \partial_\tau^n \mathbf{F}(l_1, l_2, m_1, m_2, 0)\hat{\omega}(\mathbf{m})dm_1 dm_2. \tag{B.22}$$

Furthermore, we notice that $\partial_\tau^n \mathbf{F}(l_1, l_2, m_1, m_2, 0)$ is separable. That is, in each component of $\mathbf{F}$, we can separate the $\mathbf{l}$ and $\mathbf{m}$ dependence as follows:

$$\frac{i}{(2\pi)^2}\partial_\tau^n F_1(l_1, l_2, m_1, m_2, 0) = \frac{i}{(2\pi)^2}\sum_{\substack{r,q \\ r+q=n}} c_1^{r,q}(l_1, l_2) m_1^r m_2^q \tag{B.23}$$

and

$$\begin{aligned}\frac{i}{(2\pi)^2}\partial_\tau^n F_2(l_1,l_2,m_1,m_2,0) &= \frac{i}{(2\pi)^2}\sum_{\substack{r,q\\ r+q=n}} c_2^{r,q}(l_1,l_2)m_1^r m_2^q\\ &= -\frac{i}{(2\pi)^2}\sum_{\substack{r,q\\ r+q=n}} c_1^{r,q}(l_2,l_1)m_2^r m_1^q\\ &= -\frac{i}{(2\pi)^2}\sum_{\substack{r,q\\ r+q=n}} c_1^{r,q}(l_2,l_1)m_1^q m_2^r,\end{aligned}\tag{B.24}$$

since $F_2(l_1,l_2,m_1,m_2,\tau) = -F_1(l_1,l_2,m_1,m_2,\tau)$.

Equations (B.23) and (B.24) give the form of the differential operator in (B.17). For example, the first two terms are given by

$$c_1^{0,0}(\mathbf{l}) = -\frac{l_2}{l_1^2+l_2^2},\tag{B.25}$$

$$c_1^{0,1}(\mathbf{l}) = \frac{l_1^2-l_2^2}{(l_1^2-l_2^2)^2} \quad\text{and}\quad c_1^{1,0}(\mathbf{l}) = -\frac{2l_1l_2}{(l_1^2+l_2^2)^2}.\tag{B.26}$$

Now, we use the formulae (B.23), (B.24) in the expansion (B.22) for I to get

$$I_j = \sum_{n=0}^{\infty}\frac{\tau^n}{n!}\sum_{\substack{r,q\\ r+q=n}} c_j^{r,q}(\mathbf{l})\frac{i}{(2\pi)^2}\int\int_{\mathbf{R}^2\backslash B} m_1^r m_2^q\hat\omega(\mathbf{m})dm_1dm_2,\tag{B.27}$$

where $j = 1,2$ identifies the component. We want to identify the moments of $\hat\omega$ (in (B.27)) with derivatives of ω. To do this however, we need to extend the region of integration to all $\mathbf{R}^2$. We must also analyze the error in doing this. Write,

(B.28)

$$\begin{aligned}&\frac{i}{(2\pi)^2}\int\int_{\mathbf{R}^2\backslash B} m_1^r m_2^q\hat\omega(\mathbf{m})dm_1dm_2\\ &\quad= \frac{i}{(2\pi)^2}\int\int_{\mathbf{R}^2} m_1^r m_2^q\hat\omega(\mathbf{m})dm_1dm_2 - \frac{i}{(2\pi)^2}\int\int_{B} m_1^r m_2^q\hat\omega(\mathbf{m})dm_1dm_2\\ &\quad= i^{1-r-q}[\partial_x^r\partial_y^q\omega](\mathbf{0}) - \frac{i}{(2\pi)^2}\int\int_{B} m_1^r m_2^q\hat\omega(\mathbf{m})dm_1dm_2.\end{aligned}$$

The error term (the second term of (B.28)) can be seen to be of arbitrary order since $\mathbf{l}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}$. This is shown as follows: Since $|\hat\omega(\mathbf{m})| \le \frac{c}{|\mathbf{m}|^M}$ for any M with $|\mathbf{m}|$ sufficiently large (because ω is smooth), we get

(B.29)

$$\begin{aligned}E \equiv \left|\int\int_B m_1^r m_2^q\hat\omega(\mathbf{m})dm_1dm_2\right| &\le c\int\int_B |\mathbf{m}|^n\frac{1}{|\mathbf{m}|^M}dm_1dm_2\\ &\le c\int\int_B |\mathbf{m}|^{n-M}dm_1dm_2\\ &\le c\tau^{M-n}\int_-^+\int_-^+\frac{d\tilde m_1 d\tilde m_2}{[(\tau\tilde m_1+l_1)^2+(\tau\tilde m_2+l_2)^2]^{\frac{M-n}{2}}}\end{aligned}$$

by changing variables. Now, let us suppose that $\mathbf{l} = (l_1, l_2) \in \Lambda^*(\alpha_1, \alpha_2)\backslash\mathbf{0}$. Then, we can choose $\tau(= \frac{h}{2\pi})$ small enough

$$\text{(B.30)} \qquad \tau < \tfrac{1}{4}\min(1/\alpha_1, 1/\alpha_2),$$

so that there is no singularity in the integrand of (B.29), and obtain the bound

$$\text{(B.31)} \qquad E \leq c\frac{(2\tau)^{M-n}}{(l_1^2 + l_2^2)^{\frac{M-n}{2}}}.$$

As this is true for any M, this term is of arbitrary order. Therefore, combining (B.27), (B.28), (B.31), we get

$$\text{(B.32)} \qquad I_j = \sum_{n=0}^{N-1} \frac{\tau^n}{n!} \sum_{\substack{r,q \\ r+q=n}} c_j^{r,q}(\mathbf{l}) i^{1-r-q}[\partial_x^r \partial_y^q \omega](\mathbf{0}) + O\left(\frac{\tau^N}{|l|^N}\right)$$

for any N and $\mathbf{l} \in \Lambda^*(\alpha_1, \alpha_2)\backslash\mathbf{0}$. This takes care of the first term in (B.21).

It now remains to consider the second term.

$$\text{(B.33)} \qquad II = \frac{i}{(2\pi)^2}\int\int_B \mathbf{F}(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1 dm_2.$$

This term is also seen to be of arbitrary order.

Without loss of generality, we consider the case with F_1. We have

$$\text{(B.34)} \qquad II_1 = \frac{i}{(2\pi)^2}\int\int_B F_1(l_1, l_2, m_1, m_2, \tau)\hat{\omega}(\mathbf{m})dm_1 dm_2.$$

By changing variables and using again that $|\hat{\omega}(m)| \leq \frac{c}{|\mathbf{m}|^M}$, for any M, we obtain

$$\text{(B.35)} \qquad |II_1| \leq c\tau^{-2}\int_{-\tau}^{+\tau}\int_{-\tau}^{+\tau} \frac{|\tilde{m}_2|}{\tilde{m}_1^2 + \tilde{m}_2^2} \frac{d\tilde{m}_1 d\tilde{m}_2}{\left[\left(\frac{\tilde{m}_2 + l_2}{\tau}\right)^2 + \left(\frac{\tilde{m}_1 + l_1}{\tau}\right)^2\right]^{\frac{M}{2}}}.$$

Now, if $\mathbf{l} \in \Lambda^*(\alpha_1, \alpha_2)\backslash\mathbf{0}$ and τ satisfies (B.30), then the only singularity of the integrand is due to the term $1/(\tilde{m}_1^2 + \tilde{m}_2^2)$, which is integrable. Consequently, we have

$$\text{(B.36)} \qquad |II_1| \leq c\frac{(2\tau)^{M-1}}{(l_1^2 + l_2^2)^{M/2}},$$

and thus this term is also of arbitrary order.

Now, combining the results, (B.14), (B.21), (B.32), and (B.36), we get the result

(B.37)

$$\text{P.V.} \sum_{l_\mathbf{j} \in \Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} (\widehat{\mathbf{K}\omega})_m\left(\frac{1}{\tau}\mathbf{l_j}\right) = -\sum_{n=0}^{N-1}\frac{\tau^{n+1}}{n!}\lim_{\epsilon \to o}\sum_{l_\mathbf{j} \in \Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \hat{\delta}_\epsilon\left(\frac{1}{\tau}\mathbf{l_j}\right)$$
$$\cdot \sum_{\substack{r,q \\ r+q=n}} c_m^{r,q}(\mathbf{l_j}) i^{1-r-q}[\partial_x^r \partial_y^q \omega](\mathbf{0}) + O(\tau^{N+1})$$

for any N sufficiently large, and where $m = 1, 2$ denotes the component.

Finally, the $n = 0$ term, all the terms even in n, and parts of the odd terms vanish due to oddness of the summands in the lattice sum. (See (B.25), (B.26) for example.) Therefore, identifying $\tau = h/2\pi$ and the surviving terms with $\tilde{\mathrm{C}}(\alpha)[\omega]$ yields the result (B.16), and thus completes the proof of the Fourier lattice sum expansion lemma. This lemma, combined with the self-induction term, proves the result (A.6) and completes the proof of the point vortex error expansion.

Appendix C. Calculating the Second-Order Coefficient. In this appendix, we derive the expression in (2.8) for the second-order error coefficient $C[\omega]_{j,2}$. Although the value of the second-order term is independent of the choice of smoothing function used, the ease with which it can be actually computed depends very much on this choice. The higher-order coefficients present no such difficulty as their associated lattice sums converge absolutely and can thus be calculated (if necessary) in a straightforward manner.

We begin with some notation. Define the lattice sum by

$$S(\alpha) = 4\sum_{m=1}^{\infty} \frac{1}{m}[f(m;\alpha)], \tag{C.1}$$

where

$$f(m;\alpha) = \frac{1}{m}\sum_{l=0}^{m}{}' \left[\frac{(l/m)^2-\alpha^2}{[(l/m)^2+\alpha^2]^2} + \frac{1-\alpha^2(l/m)^2}{[1+\alpha^2(l/m)^2]^2}\right], \tag{C.2}$$

and $\sum'$ denotes the trapezoidal sum(i.e., the endpoints are weighted by $\frac{1}{2}$). The purpose of this appendix is to prove the following result given in §2.

THEOREM. SECOND-ORDER ERROR COEFFICENT. *For $\omega(\mathbf{x})$ smooth and rapidly decaying, the second-order error coefficient can be given as*

$$\mathrm{C}[\omega]_{\mathrm{j},2}h^2 = \frac{1}{4\pi}(-\omega_y(\mathbf{x_j})C_2(\alpha), \omega_x(\mathbf{x_j})C_2(1/\alpha))h_1h_2, \tag{C.3}$$

where $h_1 = \alpha_1 h$, $h_2 = \alpha_2 h$, α is the grid aspect ratio α_1/α_2, and

$$C_2(\alpha) = \frac{4}{\pi}\arctan(1/\alpha) - \frac{\alpha}{\pi}S(\alpha). \tag{C.4}$$

Remark 1. In the special case of a square grid $\alpha = 1$, we have $S(1) = 0$ and the second-order term then has the form

$$\mathrm{C}[\omega]_{\mathrm{j},2}h^2 = \frac{1}{4\pi}\nabla^{\perp}\omega(\mathbf{x_j})h_1h_2,$$

where $\nabla^{\perp}\omega = (-\omega_{x_2}, \omega_{x_1})$.

Remark 2. In the analysis of Goodman, Hou, and Lowengrub [16], it is essentially the smoothing function that gives rise to the undetermined constant of integration appearing in their second-order coefficient. We determine it explicitly.

Proof. Without loss of generality, we consider the case $\mathbf{x}_j = \mathbf{0}$. From (B.6), (B.9), (B.14), (B.26), and (B.27) $\mathrm{C}[\omega]_{0,2}$ can be given as

(C.5)

$$\mathrm{C}[\omega]_{0,2}h^2 =$$

$$\lim_{\epsilon\to 0}(\mathbf{K}w)_\epsilon(\mathbf{0})h_1h_2 - \nabla^{\perp}w(\mathbf{0})\left(\frac{h}{2\pi}\right)^2 \lim_{\epsilon\to 0}\sum_{\mathbf{l}\in\Lambda^*(\alpha_1,\alpha_2)\backslash\mathbf{0}} \frac{l_1^2-l_2^2}{|\mathbf{l}|^4}\hat{\delta}_\epsilon\left(\frac{2\pi}{h}\mathbf{l}\right),$$

where $\mathbf{l} = (l_1, l_2)$ and $h_1 = \alpha_1 h, h_2 = \alpha_2 h$.

The two terms to be considered are the self-induction term

$$\lim_{\epsilon\to 0} (\mathbf{K}\omega)_\epsilon(\mathbf{0}) \tag{C.6}$$

and the lattice sum

$$\lim_{\epsilon\to 0} \sum_{\mathbf{l}\in\Lambda^*(\alpha_1,\alpha_2)\setminus\mathbf{0}} \frac{l_1^2 - l_2^2}{(l_1^2 + l_2^2)^2} \hat{\delta}_\epsilon\left(\frac{2\pi}{h}\mathbf{l}\right). \tag{C.7}$$

Setting $\epsilon = 0$ in (C.7) gives a sum that is only conditionally convergent, and whose value depends upon how it is summed. The specific summation rule is dictated by $\hat{\delta}_\epsilon$. Here, we choose a sharp square cutoff. Let $\epsilon = \frac{1}{M}$ and

$$\hat{\delta}_{\frac{1}{M}}\left(\frac{2\pi}{h}\mathbf{l}\right) = \begin{cases} 1 \text{ if } |l_1| \le \dfrac{M}{\alpha_1} \text{ and } |l_2| \le \dfrac{M}{\alpha_2}, \\ 0 \text{ otherwise.} \end{cases} \tag{C.8a}$$

This in turn gives the Dirichlet kernel

$$\delta_{\frac{1}{M}}(x_1, x_2) = \frac{1}{\pi^2} \frac{\sin\left(\dfrac{Mx_1}{\alpha_1}\right)}{x_1} \frac{\sin\left(\dfrac{Mx_2}{\alpha_2}\right)}{x_2}. \tag{C.8b}$$

Now, using (C.8a) in (C.7) gives the lattice sum

$$\alpha\alpha_1\alpha_2 \lim_{M\to\infty} \sum_{l_1=-M}^{+M} \sum_{\substack{l_2=-M \\ l\neq 0}}^{+M} \frac{l_1^2 - \alpha^2 l_2^2}{(l_1^2 + \alpha^2 l_2^2)^2}, \tag{C.9}$$

and using (C.8b) in (C.6) yields

(C.10)

$$\lim_{M\to\infty} (\mathbf{K}\omega)_{\frac{1}{M}}(\mathbf{0}) = \frac{1}{\pi^2} \lim_{M\to\infty} \int\int_{\mathbf{R}^2} \mathbf{K}(\mathbf{x})\omega(\mathbf{x}) \frac{\sin(Mx_1/\alpha_1)}{x_1} \frac{\sin(Mx_2/\alpha_2)}{x_2} dx_1 dx_2.$$

Thus, we need to determine (C.9) and (C.10). We begin with the self-induction term (C.10).

SELF-INDUCTION LEMMA. *For $\omega(\mathbf{x})$ smooth and rapidly decaying, the self-induction term is given by*

$$\lim_{M\to\infty} (\mathbf{K}\omega)_{\frac{1}{M}}(\mathbf{0}) = \frac{1}{\pi^2}\left(-\omega_{x_2}(0) \arctan\left(\frac{1}{\alpha}\right), \omega_{x_1}(0) \arctan(\alpha)\right). \tag{C.11}$$

Proof. Apply (A.19) with $f = K_i\omega$, where $i = 1, 2$ denotes the component of $\mathbf{K}$. In light of (A.18), (A.20)–(A.22), and (A.28), it is clear that the proof reduces to evaluating

(C.12)

$$\lim_{M\to\infty} (K_i\omega)_{\frac{1}{M}}(\mathbf{0}) = \lim_{M\to\infty} \frac{1}{\pi^2} \nabla\omega(\mathbf{0})$$
$$\cdot \int_{\frac{-RM}{\alpha_1}}^{\frac{+RM}{\alpha_1}} \int_{\frac{-RM}{\alpha_2}}^{\frac{+RM}{\alpha_2}} \tilde{\mathbf{H}}_i(\alpha_1 x_1, \alpha_2 x_2) \frac{\sin(x_1)}{x_1} \frac{\sin(x_2)}{x_2} dx_2 dx_1,$$

where $\tilde{\mathbf{H}}_i(\mathbf{x}) = \mathbf{x}K_i(\mathbf{x})$. Without loss of generality, take $i = 2$. Then, (C.12) reduces to (C.13)

$$\lim_{M\to\infty}(K_2\omega)_{\frac{1}{M}}(\mathbf{0}) = \frac{\omega_{x_1}(\mathbf{0})}{\pi^2}\lim_{M\to\infty}\int_{\frac{-RM}{\alpha_1}}^{\frac{+RM}{\alpha_1}}\int_{\frac{-RM}{\alpha_2}}^{\frac{+RM}{\alpha_2}}\frac{x_1^2}{x_1^2+\left(\frac{x_2}{\alpha}\right)^2}\frac{\sin(x_2)}{x_2}\frac{\sin(x_1)}{x_1}dx_2dx_1,$$

where $\alpha = \alpha_1/\alpha_2$ is the grid aspect ratio. The term involving ω_{x_2} vanishes due to oddness of the integrand in each coordinate direction. With some care, the inner integral can be evaluated by contour integration. In particular, the regions of integration of both the inner and outer integrals must be divided appropriately so that the contour integration is justified. We do not present the details here. The K_1 integral is evaluated similarly. It now remains to consider the lattice sum (C.9). We obtain the following result.

LATTICE SUM LEMMA. *The sum in* (C.9), *omitting the $\alpha\alpha_1\alpha_2$ prefactor, is given by* (i)

$$\text{(C.14)}\qquad S(\alpha) = \lim_{M\to\infty}\sum_{l_1=-M}^{+M}\sum_{\substack{l_2=-M\\ l\neq o}}^{+M}\frac{l_1^2-\alpha^2l_2^2}{(l_1^2+\alpha^2l_2^2)^2} = 4\sum_{m=1}^{\infty}\frac{1}{m}f(m;\alpha),$$

where

$$\text{(C.15)}\qquad f(m;\alpha) = \frac{1}{m}\sum_{k=0}^{m}{}'\left[\frac{(k/m)^2-\alpha^2}{[(k/m)^2+\alpha^2]^2}+\frac{1-\alpha^2(k/m)^2}{[1+\alpha^2(k/m)^2]^2}\right],$$

where $\sum'$ denotes the trapezoidal sum (i.e., the endpoints are weighted by $\frac{1}{2}$). For $m \gg 1$, we have (ii)

$$\text{(C.16)}\qquad \begin{aligned} f(m;\alpha) &= \frac{1}{6}\frac{\alpha^2-1}{(1+\alpha^2)^2}\frac{1}{m^2} - \frac{1}{30}\frac{(\alpha^2-1)(\alpha^4-10\alpha^2+1)}{(1+\alpha^2)^4}\frac{1}{m^4} + O\left(\frac{1}{m^6}\right)\\ &\equiv f_2(\alpha)\frac{1}{m^2} + f_4(\alpha)\frac{1}{m^4} + O\left(\frac{1}{m^6}\right). \end{aligned}$$

Proof of Lemma. (i) Using the four-fold symmetry of the summand over quadrants, we rewrite the sum (C.9) as

$$\text{(C.17)}\quad S(\alpha) = \lim_{M\to\infty}\sum_{l_1=-M}^{+M}\sum_{\substack{l_2=-M\\ l\neq 0}}^{+M}\frac{l_1^2-\alpha^2l_2^2}{(l_1^2+\alpha^2l_2^2)^2} = \lim_{M\to\infty}4\sum_{l_1=0}^{M}{}''\sum_{\substack{l_2=0\\ l\neq 0}}^{M}{}''\frac{l_1^2-\alpha^2l_2^2}{(l_1^2+\alpha^2l_2^2)^2},$$

where $\sum''$ denotes that terms with $l_1 = 0$ or $l_2 = 0$ are weighted by $\frac{1}{2}$. Then, by summing first out along the diagonal of the lattice, we get

$$\text{(C.18)}\qquad S(\alpha) = \lim_{M\to\infty}4\sum_{m=1}^{M}\sum_{k=0}^{m}{}'\left[\frac{k^2-\alpha^2m^2}{(k^2+\alpha^2m^2)^2}+\frac{m^2-\alpha^2k^2}{(m^2+\alpha^2k^2)^2}\right],$$

$$\text{(C.19)}\qquad = \lim_{M\to\infty}4\sum_{m=1}^{M}\frac{1}{m}\left[\frac{1}{m}\sum_{k=0}^{m}{}'\frac{(k/m)^2-\alpha^2}{((k/m)^2+\alpha^2)^2}+\frac{(1-\alpha^2(k/m)^2)^2}{(1+\alpha^2(k/m)^2)^2}\right],$$

$$= 4\sum_{m=1}^{\infty} \frac{1}{m} f(m;\alpha), \tag{C.20}$$

which gives the result (i).

The self-induction lemma and result (i) of the lattice sum lemma together give the second-order error coefficient in (C.3) and (C.4), and completes the proof of the theorem. We now consider result (ii) of the lattice sum lemma, which gives the large m asymptotics of f and leads to our rapid summation corollary.

The function $f(m;\alpha)$, (C.15), is the trapezoidal rule approximation of the integrals

$$\int_0^1 \frac{x^2-\alpha^2}{(x^2+\alpha^2)^2}dx + \int_0^1 \frac{1-\alpha^2x^2}{(1+\alpha^2x^2)^2}dx. \tag{C.21}$$

Thus, $f(m;\alpha)$ can be written as

$$f(m;\alpha) = \int_0^1 \frac{x^2-\alpha^2}{(x^2+\alpha^2)^2}dx + \int_0^1 \frac{1-\alpha^2x^2}{(1+\alpha^2x^2)^2}dx + E_1(m;\alpha) + E_2(m;\alpha), \tag{C.22}$$

where the errors E_1 and E_2 are given by

$$E_1(m;\alpha) = \frac{1}{m}\sum_{k=0}^{m}{}' \frac{(k/m)^2-\alpha^2}{((k/m)^2+\alpha^2)^2} - \int_0^1 \frac{x^2-\alpha^2}{(x^2+\alpha^2)^2}dx \tag{C.23}$$

and

$$E_2(m;\alpha) = \frac{1}{m}\sum_{k=0}^{m}{}' \frac{1-\alpha^2(k/m)^2}{(1+\alpha^2(k/m)^2)^2} - \int_0^1 \frac{1-\alpha^2x^2}{(1+\alpha^2x^2)^2}dx. \tag{C.24}$$

Now, by direct calculation,

$$\int_0^1 \frac{x^2-\alpha^2}{(x^2+\alpha^2)^2}dx + \int_0^1 \frac{1-\alpha^2x^2}{(1+\alpha^2x^2)^2}dx = 0. \tag{C.25}$$

And, for $m \gg 1$, the error terms can be found through the Euler–MacLaurin expansion. This gives the result (ii).

We can now rapidly and accurately sum $S(\alpha)$ using the asymptotic form of f. In particular, we obtain the following corollary.

RAPID SUMMATION COROLLARY. *The sum $S(\alpha)$ can be rewritten so as to converge like $O(1/m^7)$ by using the expansion* (ii) *in the following way*:

$$S(\alpha) = \sum_{m=1}^{\infty} \frac{1}{m}\left[f(m;\alpha) - f_2(\alpha)\frac{1}{m^2} - f_4(\alpha)\frac{1}{m^4}\right] + f_2(\alpha)\zeta(3) + f_4(\alpha)\zeta(5), \tag{C.26}$$

where

$$\zeta(s) = \sum_{m=1}^{\infty} \frac{1}{m^s}, \qquad s > 1,$$

is the Riemann–Zeta function. In particular, $\zeta(3) = 1.202056903159594$ *and* $\zeta(5) = 1.036927755143370$.

Acknowledgments. We thank Chris Anderson, Jonathan Goodman, Tom Hou, Joe Keller, Omar Knio, and Vered Rom-Kedar for valuable discussions. We thank the referees for their careful examination of the paper and their useful comments. We especially thank one of the referees for noticing a problem with our original formulation of the Poisson summation formula for singular functions. MJS would also like to thank the Mathematics Department at UCLA, where some of the writing and work on this paper was done, for their hospitality. Likewise, JSL would like to thank the Courant Institute for their hospitality.

REFERENCES

[1] C. Anderson, *A method of local corrections for computing the velocity field due to a distribution of vortex blobs*, J. Comput. Phys., 62 (1986), pp. 111–123.

[2] C. Anderson and C. Greengard, *On vortex methods*, SIAM J. Numer. Anal., 22 (1985), pp. 413–440.

[3] G. Baker and M. Shelley, *On the connection between thin vortex layers and vortex sheets*, J. Fluid Mech., 215 (1990), pp. 161–194.

[4] J. T. Beale, *A convergent* 3-D *vortex method with grid free stretching*, Math. Comp., 46 (1986), pp. 401–424.

[5] J. T. Beale and A. Majda, *High order accurate vortex methods with explicit velocity kernels*, J. Comput. Phys., 58 (1985), pp. 188–208.

[6] ———, *Vortex methods* II: *High order accuracy in* 2 *and* 3 *dimensions*, Math. Comp., 32 (1982), pp. 29–52.

[7] J. B. Bell, P. Collela, and H. M. Glaz, *A second order projection method for the incompressible Navier–Stokes equations*, J. Comput., Phys., 58 (1989), pp. 188–208.

[8] J. U. Brackbill, *Coordinate System Control: Adaptive Meshes*, in Numerical Grid Generation, J. F. Thompson, ed., Elsevier, Amsterdam, North-Holland, 1982.

[9] J. Buntine and D. I. Pullin, *Merger and cancellation of strained vortices*, J. Fluid Mech., 205 (1989), p. 1632.

[10] A. J. Chorin, *A numerical study of slightly viscous flow*, J. Fluid Mech., 57 (1973), pp. 785–796.

[11] P. Collela and P. Woodward, *The piecewise parabolic method*, J. Comput. Phys., 54 (1984), p. 174.

[12] G. H. Cottet, *Methodes Particularies pour l'Equation d'Euler dans le Plan*, Thèse 3eme Cycle, Univ. Pierre et Marie Curie, Paris, France, 1982.

[13] G. H. Cottet, J. Goodman, and T. Y. Hou, *Convergence of the grid-free point vortex method for the three-dimensional Euler equations*, SIAM J. Numer. Anal., 28 (1991), pp. 291–307.

[14] H. Dym and H. P. McKean, *Fourier Series and Integrals*, Academic Press, New York, 1972.

[15] D. Fishelov, *A new vortex scheme for viscous flow*, J. Comput. Phys., 26 (1990), p. 399.

[16] J. Goodman, T. Y. Hou, and J. Lowengrub, *Convergence of the point vortex method for the* 2-D *Euler equations*, Comm. Pure Appl. Math., 43 (1990), pp. 415–430.

[17] L. Greengard and V. Rokhlin, *A fast algorithm for particle summations*, J. Comput. Phys., 73 (1987), pp. 325–348.

[18] O. Hald, *Convergence of vortex methods* II, SIAM J. Numer. Anal., 16 (1979), pp. 726–755.

[19] A. Harten and S. Osher, *Uniformly high order accurate non-oscillatory schemes*, I, SIAM J. Numer. Anal., 24 (1987), pp. 279–309.

[20] T. Y. Hou, J. Lowengrub, and M. J. Shelley, *The convergence of an exact desingularization and local regridding for vortex methods*, SIAM J. Sci. Comput., 14 (1993), pp. 1–18.

[21] J. Lowengrub and M. J. Shelley, *Smooth Grid Methods for Vorticity Formulation of the Euler Equations*, in Vortex Dynamics and Methods, C. Anderson and C. Greengard, eds., Lectures in Applied Mathematics 18, American Mathematical Society, Providence, RI, 1991.

[22] R. B. Payne, *Calculations of unsteady viscous flow past a circular cylinder*, J. Fluid Mech., 4 (1958), pp. 81–86.

[23] R. Peyret and T. Taylor, *Computational Methods for Fluid Flow*, Springer Series in Computational Physics, Springer-Verlag, Berlin, New York, 1983.

[24] P. A. Raviart, *An Analysis of Particle Methods*, in Numerical Methods in Fluid Dynamics, Lecture Notes in Math. 1127, Springer-Verlag, Berlin, New York, 1985, p. 243.

[25] V. Rom-Kedar and M. J. Shelley, *The dynamics of plumes in continuously stratified fluids*, manuscript.

[26] L. Rosenhead, *The point vortex approximation of a vortex sheet*, Proc. Roy. Soc. London Ser. A, 134 (1932), pp. 170–192.

[27] M. J. Shelley, D. Meiron, and S. Orszag, *Dynamical aspects of vortex reconnection of perturbed anti-parallel vortex tubes*, J. Fluid Mech., 246 (1993), p. 613.

[28] M. J. Shelley, *A study of singularity formation in vortex sheet motion by a spectrally accurate vortex method*, J. Fluid Mech., 244 (1992), p. 493.

[29] A. Sidi and M. Israeli, *Quadrature methods for singular and weakly Fredholm integral equations*, J. Sci. Comput., 3 (1988), pp. 201–231.

[30] E. M. Stein and G. Weiss, *Introduction to Fourier Analysis on Euclidean Spaces*, Princeton University Press, Princeton, NJ, 1971.

[31] J. Stoer and R. Burlirsch, *Introduction to Numerical Analysis*, Springer-Verlag, Berlin, New York, 1980.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1143–1158, September 1993

HYPERCUBE ALGORITHMS FOR DIRECT N-BODY SOLVERS FOR DIFFERENT GRANULARITIES*

JEAN-PHILIPPE BRUNET[†], JILL P. MESIROV[†], AND ALAN EDELMAN[‡]

Abstract. Algorithms for the N-body problem are compared and contrasted, particularly those where N is in the range for which direct methods outperform approximation methods. With fewer bodies than processors, the so-called "replicated orrery" on a three-dimensional grid has been used successfully on the Connection Machine CM-2 architecture. With more bodies, the "rotated and translated Gray codes" is an ideal direct algorithm for machines such as the CM-2 in that it takes optimal advantage of the communications bandwidth of the machine.

A classical Latin square can be used to abstractly denote any direct N-body calculation. Computational windows superimposed on this square illustrate the granularity of the computation. This point of view naturally illustrates a sequence of algorithms ranging along a granularity scale in the following order: "massively parallel," "replicated orrery," "orrery," "rotated/translated Gray codes," and "serial."

Key words. N-body, hypercube, CM-2, parallel computing, Latin square, Gray code

AMS subject classifications. 70-80

1. Introduction.

In a wide variety of applications areas, including astronomy [5], [18], [29]; molecular dynamics [17], [27], [32], [33]; and fluid dynamics [16], [25], [30], [31], a serious computational bottleneck occurs from the need to determine the mutual interaction of N bodies. Directly computing the N^2 pairwise interactions given by

$$\text{Force(body } i) = \sum_j \text{Force(body } i \text{ due to body } j)$$

on a conventional computer can take an unacceptably long amount of time for values of N which are exactly in the range of interest for the application.

Approximation methods are one approach towards reducing computational complexity. Examples of this are the particle/mesh [21], local correction [1], hierarchical [2], [7], and multipole [15] methods. The structure of such algorithms has been investigated by Katzenelson [24].

Another approach is to keep to the naive direct method, but to design algorithms that take advantage of advanced computer architectures like the Connection Machine CM-2. We have chosen to take this path and to explore only direct methods in this work. Although this approach has asymptotically higher complexity, these algorithms remain useful for two important reasons: 1) they are simpler to code and maintain thus minimizing the important commodity of programming effort, and 2) they give competitive performance for a relevant range of N. Though the approximation algorithms must ultimately be advantageous for large values of N, for a number of problems of physical interest, N is in the range where direct N-body algorithms are superior. See [30] for further discussion of the advantages of a direct algorithm over an approximation algorithm.

The algorithms for direct N-body solvers that we will discuss in this paper are designed for use on a parallel architecture with a hypercube network topology. In fact, we

*Received by the editors November 13, 1991; accepted for publication (in revised form) September 30, 1992.

[†]Thinking Machines Corporation, 245 First Street, Cambridge, Massachusetts 02142 (brunet@think.com, mesirov@think.com).

[‡]Department of Mathematics, University of California, Berkeley, California 94720 (edelman @math.berkeley.edu).

feel that hypercube architectures are ideally suited to the direct method. For nondirect methods on the hypercube, see [4], [14], [19], and [34].

We focus on two different algorithms, but we take a point of view that we have not yet seen in the literature: We show that the two algorithms fit along a continuous spectrum. Perhaps we can make an analogy with the notion that X-rays and radio waves properly fit along one axis. However, instead of using frequency, the important parameter here is *granularity* measured by the ratio P/N.

For the case where the number of processors P satisfies $N \leq P \leq N^2$, we describe an optimal mapping of a family of *replicated orreries* onto a hypercube. For the case where $P < N$, we present a multiwire all-to-all broadcast algorithm which utilizes "rotated and translated Gray codes" to make optimal use of the communication bandwidth of the hypercube. Thus, in this approach, the communication time of the algorithm is negligible for large values of N.

Both of these algorithms have been implemented on the Connection Machine CM-2. There are two programming models for the CM-2 and each algorithm lends itself more naturally to one of them. The older model (which is becoming more and more obsolete) is known as the "fieldwise model," while the more recent model is the "slicewise model." From a conceptual point of view, the value of P is 32 times greater for the fieldwise model than it is for the slicewise model. Both approaches, and thus both programming models, were used for the N-body solver in the implementation of a random vortex method for numerical simulation of fluid flow [30]; we will analyze the computational complexity of each approach.

Section 2 discusses N-body problems in an abstract mathematical context. In §§3 and 4 we present two concrete algorithms with complexity analyses: the replicated orrery and a multiwire algorithm. The CM-2 implementation of both algorithms is briefly described in §5. A sample application is described in §6 and the performance of the CM-2 implementation of both approaches is discussed.

2. Latin squares and N-body problems.

Before we analyze in detail two particular approaches to solving direct N-body problems that have been successful, we would like to abstract out the important mathematical features of an N-body computation on a parallel machine. In particular, the two important questions are *where* (which processor?) and *when* (which timestep?) does each interaction occur. By timestep we mean those taken in calculating the forces for a given particle configuration rather than those taken to move the particle positions.

Implicit in the knowledge of the "where and when" of the computation is the communication that occurs. Since we know which data needs to be where at a particular moment, the only degree of freedom might be how soon the data arrives at a particular processor before a computation starts.

As a mathematical abstraction, we consider it enlightening to display the "where and when" of an N-body computation on a *Latin square*. The basic idea is quite simple, however, almost any N-body problem on any parallel machine can be defined as a Latin square.

A *Latin square* is an N by N table of entries $1, 2, \ldots, N$ with the property that each number i appears exactly once in each column and once in each row. Perhaps the earliest known reference is a puzzle requiring that the 16 lettered cards (A,K,Q,J) of an ordinary 52-card deck be arranged so that no letter appears twice in the same column or row nor along a diagonal [13]. Classically, Latin squares have had many uses including the design of experiments, displays of multiplication tables of finite groups, and other recreational puzzles (see [9] and [13]). Their first systematic study was performed by Euler [13].

As an example, Fig. 1 displays a Latin square of order 4. In the context of an N-body computation, the entry a in row i and column j can symbolize a force law interaction of body i with body j. Thus in Fig. 1, the first column depicts the interactions of body 1 with bodies 2, 3, 4, and 1, respectively. We remark that in practice, self-interactions are usually omitted. Also in practice, symmetric interactions may be avoided, giving a speedup of a factor at most equal to 2.

Body:	1	2	3	4
	2	3	4	1
	3	4	1	2
	4	1	2	3
	1	2	3	4

FIG. 1. *A Latin square of order 4. To indicate an* N-*body calculation, we let the entry in row i and column j denote an interaction between that body and body j. A serial algorithm could proceed from left to right and from top to bottom along this square, one box at a time.*

In the sections to follow, we study the five possibilities given in Fig. 2. Specifically, the replicated orrery is described in detail in §3, while the rotated and translated Gray codes are described in §4. Before we give the details of the algorithms, however, we can already describe some necessary features based on the granularity alone.

			Example		
			P	N	$O(T_{comm})$
Case I	$P = N^2$	Massively parallel	16	4	$\log N$
Case II	$N < P < N^2$	Replicated orrery	8	4	$N^2/P + \log(P/N)$
Case III	$P = N$	Orrery	4	4	N
Case IV	$1 < P < N$	Rot/trans Gray codes	2	4	$N/\log P$
Case V	$P = 1$	Serial	1	4	0

FIG. 2. *Possible granularities (Example for $N = 4$). The communications complexities are discussed in Sections 3 and 4.2. The actual communications complexity for Case IV is the order of $\lceil \frac{N}{P \log P} \rceil P$. The approximation listed in the table breaks down for P near N.*

For each of the five cases in Fig. 2, Fig. 3 exhibits the simultaneity of one parallel computation when $N = 4$. For clarity of exposition, we omit the entries of the square, but instead superimpose a computational "window" upon the square. The size of this window is exactly the number of processors. In each window, there is one interaction for each processor. The computation proceeds by sweeping the window across the square. Of course, if the number of bodies is held constant, then the size of the window grows with the number of processors, and the number of timesteps (equals the number of windows needed to cover the square) decreases.

The two extremes are given by Case I, massively parallel, and Case V, serial, in Fig. 3. In Case I, where $P = N^2$, there is only one window—all N-body computations occur in parallel. On the other hand, in Case V where $P = 1$, the algorithm is serial and only one interaction occurs in any given timestep. The processors are labeled p_i in the windows, and the computation proceeds by shifting the windows and the corresponding processors over the Latin square.

We now see that once the relationship between P and N is given, all that we need to specify a direct N-body computation is (i) a choice of Latin square and (ii) a mapping

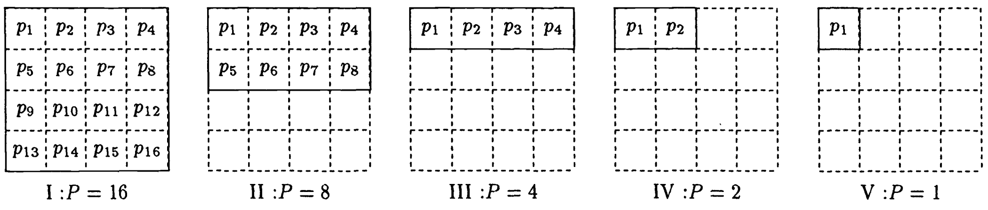


FIG. 3. *Computational windows superimposed on the Latin square ($N = 4$). The entries of the square are omitted, while the processor numbers for the first timestep of the computation are included. Later timesteps are obtained by sweeping the window across the square, so as to completely cover the square.*

from the processors to the window. We explore this in further detail in later sections, but we can give an illustrative example as we do in Fig. 4. This figure illustrates an example when $N = 8$ and $P = 4$. The square on the left is readily checked to be a Latin square. The construction of the square is quite simple. We assume that there are two bodies per node as illustrated on the two-dimensional hypercube in the figure. One copy of the bodies remains static, while one circulates; the first body circulates counterclockwise, while the second body circulates clockwise.

			body				
1	2	3	4	5	6	7	8
			interacts with				
1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
2	3	4	1	8	5	6	7
8	5	6	7	2	3	4	1
3	4	1	2	7	8	5	6
7	8	5	6	3	4	1	2
4	1	2	3	6	7	8	5
6	7	8	5	4	1	2	3

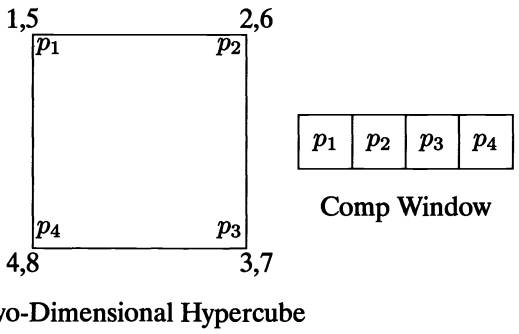


FIG. 4. *Latin square and computational windows for an 8-body problem with four processors. Each 1×4 rectangle denotes the four interactions of one timestep, one in each processor. The entries of the Latin square are obtained by circulating a dynamic copy of the first body counterclockwise, while circulating the second body clockwise on the square. A static copy of each body remains in the processor. Communication is indicated by the vertical space between pairs of lines. After the first circulation, body 1 can interact with bodies 2 and 8, while body 5 can also interact with bodies 2 and 8 without further communication. Thus a total of four windows can pass over the square before another communication need occur.*

3. $N \leq P \leq N^2$: Orreries and replicated orreries. One standard approach to N-body solvers uses N processors connected as a ring. Processor i accumulates the forces on body i. At each timestep, N of the pairwise interactions are computed, then the appropriate data are passed around the ring so that in N steps all N^2 interactions have been computed and accumulated into the correct processor; see Fig. 5. This particular parallel implementation of the N-body algorithm has been referred to as the "digital orrery" [3], as the "systolic loop" [33], and as "systolic loop double" [27].

As a Latin square, this is the addition table of the Abelian cyclic group of order N: the square is circulant, and the (i, j)th entry is essentially $i - j$ modulo n. This case is illustrated in Fig. 3 as Case III, which shows that one row of interactions is computed in each timestep.

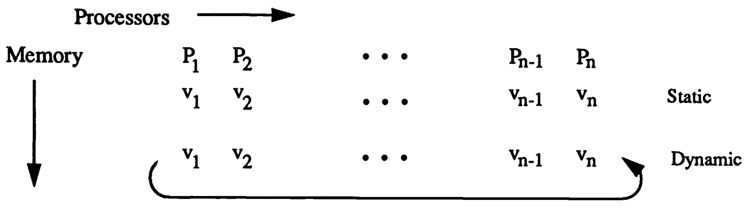

Processors
Memory
P_1 P_2 ... P_{n-1} P_n
v_1 v_2 ... v_{n-1} v_n
Static
v_1 v_2 ... v_{n-1} v_n
Dynamic

FIG. 5. *Digital orrery: $P = N$. Cycle through N shifts.*

With more than N processors one can replicate the orrery, perform a different piece of the computation on each copy, and accumulate the summands for the forces at the end [6]. For example, with $2N$ processors the computation takes place on two rings of processors (see Fig. 6), each of which passes its data only $N/2$ times. More generally, with MN processors we use M rings of N processors each. At each computation step MN interactions are computed; N/M steps with cyclic message passing of the data are required to compute the N^2 interactions. Thus the arithmetic complexity of the algorithm is $O(N/M)$. By "arithmetic complexity," we mean the number of parallel numerical computations executed by the parallel algorithm. The time of execution of a parallel algorithm is a linear combination of its arithmetic complexity and communication complexity.

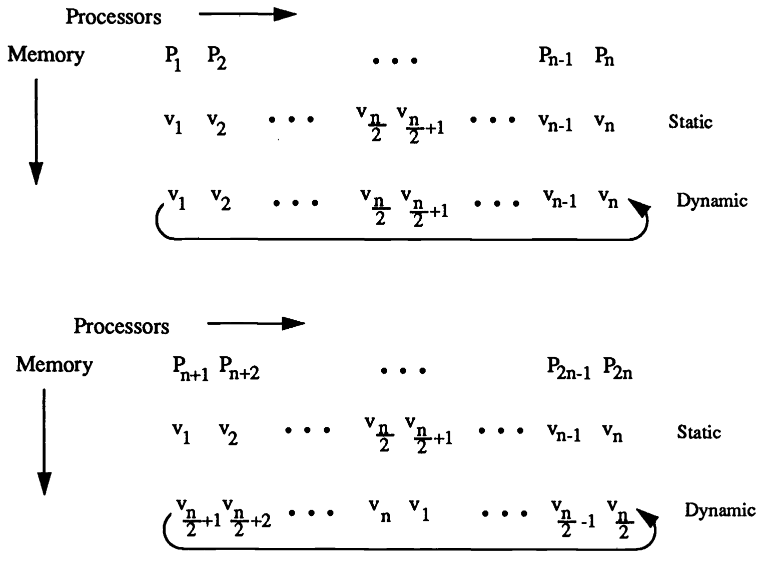

Processors
Memory
P_1 P_2 ... P_{n-1} P_n
v_1 v_2 ... $v_{\frac{n}{2}}$ $v_{\frac{n}{2}+1}$ ... v_{n-1} v_n
Static
v_1 v_2 ... $v_{\frac{n}{2}}$ $v_{\frac{n}{2}+1}$ ... v_{n-1} v_n
Dynamic
Processors
Memory
P_{n+1} P_{n+2} ... P_{2n-1} P_{2n}
v_1 v_2 ... $v_{\frac{n}{2}}$ $v_{\frac{n}{2}+1}$ ... v_{n-1} v_n
Static
$v_{\frac{n}{2}+1}$ $v_{\frac{n}{2}+2}$ ... v_n v_1 ... $v_{\frac{n}{2}-1}$ $v_{\frac{n}{2}}$
Dynamic

FIG. 6. *Replicated orrery: $P = 2N$. Cycle through $N/2$ shifts.*

An efficient processor topology comes from mapping the replicated orrery onto a three-dimensional cube of processors as observed in [26]. We assume the cube is embedded in a larger dimensional hypercube in such a way that we can distribute a value to, or sum the contents of, M processors along an axis of the cube with $O(\log M)$ communications [22].

To understand how that data mapping works, assume that the hypercube is configured as an N/M by M by M cube (see Fig. 7). The data is initially resident on the front face of the cube and is sent to the top of the cube. The data on the front face is spread back, in $\log M$ time, along the y-dimension of the cube, and the data on top is spread down along the z-dimension of the cube. The data is now distributed through the cube so that each plane parallel to the xz-axis contains a copy of the orrery. The computation proceeds by computing a pairwise interaction and doing one-dimensional cyclic message passing along the x-axis. This process is repeated N/M times until all N^2 pairwise interactions have been completed. Now the corresponding results for each individual body are accumulated, in $\log M$ time, along the y-axis back to the front face of the cube.

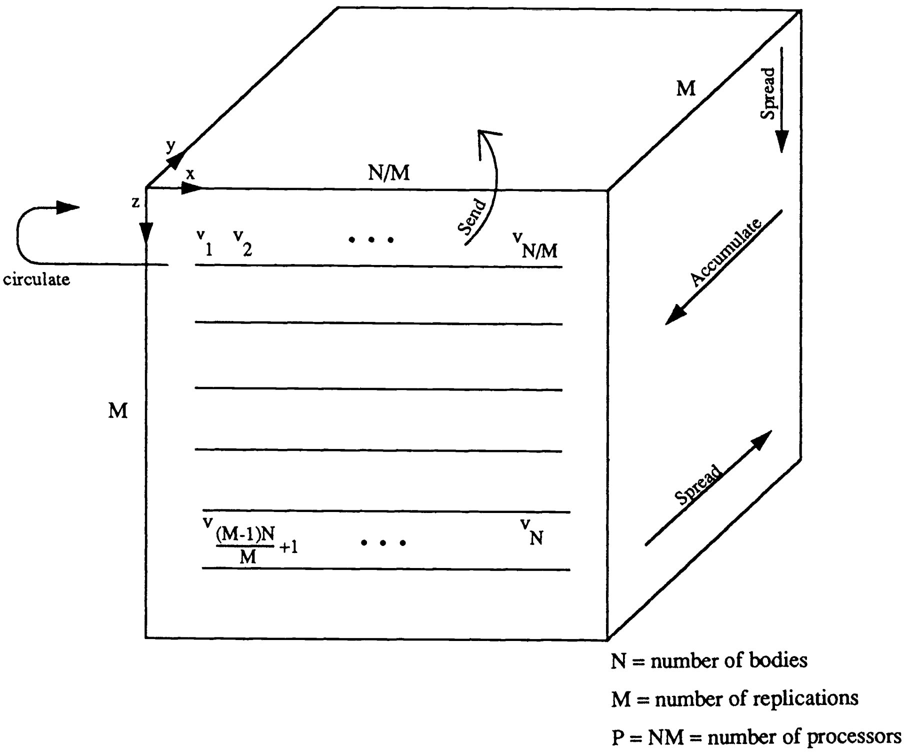

y
x
z
N/M
Send
M
Spread
circulate
v_1 v_2 • • • $v_{N/M}$
Accumulate
M
$v_{\frac{(M-1)N}{M}+1}$ • • • v_N
Spread
N = number of bodies
M = number of replications
P = NM = number of processors

FIG. 7. *Cube data mapping.*

In most cases we are restricted to M and N being powers of two. This ensures the correct wraparound for cyclic message passing, and efficient logarithmic time spreading and accumulating. Observe also that as the number of replications M increases from 1 to N, and thus the number of processors increases from N to N^2, the cube grows out from the x-axis and then projects onto the yz-plane. Thus, the dimensions of the cube can be changed to accommodate a family of parallel N-body solvers where the number of processors ranges from N to N^2. In particular, with MN processors, the solver runs in $O(N/M) + O(\log M)$ time. Note that this gives the correct complexity bounds for the extremes, i.e., $O(N)$ for the case $M = 1$, and $O(1) + O(\log N)$ when $M = N$.

Since the number of processors P is usually fixed given some particular piece of hardware, there is another useful way to view the complexity of the algorithm. Note that $P = MN$, thus, the complexity bound $O(N/M) + \log(M)$ becomes $O(N^2/P) + O(\log(P/N))$.

We remark that the algorithms described above are essentially grid algorithms. Once the data is set up, all communication is performed through cyclic message passing. Though this is the easiest conceptually, there are other approaches that make far better use of the hypercube bandwidth. In the next section, we describe a multiwire algorithm that makes optimal use of the bandwidth when $N > P$ or, to be more precise, when N is a multiple of dP, where d is the dimension of the hypercube. There are other communications primitives that make better use of the hypercube bandwidth in the orrery, such as the "multinode broadcast under the MLA assumption" described in [8] and an equivalent algorithm related to us by Ho [20]. We have chosen not to pursue this here.

Figure 8 depicts the Latin square for a replicated orrery case for $N = 8$ and $P = 16$. Note how the cube unfolds onto the computational window.

body

1	2	3	4	5	6	7	8
interacts with							
1	2	3	4	5	6	7	8
5	6	7	8	1	2	3	4
2	3	4	1	6	7	8	5
6	7	8	5	2	3	4	1
3	4	1	2	7	8	5	6
7	8	5	6	3	4	1	2
4	1	2	3	8	5	6	7
8	5	6	7	4	1	2	3

FIG. 8. *Latin square for replicated orrery*, $N = 8$, $P = 16$.

4. $P < N$: Multiwire algorithm.

4.1. The approach. In the case where $P < N$ the data mapping is much simpler. We divide the N bodies evenly over the $P = 2^d$ processors of a d-dimensional hypercube. For simplicity, assume the number of bodies in each node N/P is an integer multiple of d. To compute the N^2 interactions, the data for each body must be transmitted to all of the other processors. We want to do this using as much of the communications bandwidth of the hypercube as possible. If the processors have enough memory, then this transmission can be performed initially, and afterwards, all the arithmetic can be performed locally. This type of communication in a hypercube has been called *all-to-all broadcasting* [23] and *multinode broadcast* [8].

We do not follow this strategy exactly. We can interleave pieces of the computation with the communications and have no need to store the data for all the bodies at each node. Thus, we can use the idea of "rotated and translated Gray codes" described in [23]. Although this method has "limited potential for pipelining," it is just right for the N-body problem. A Gray code is rotated to construct as many conflict-free paths as possible leaving a node, visiting all others, and returning to its starting point. By translating the starting point of each of these paths to every node of the hypercube, the data in each node circulates to every other one in as short a time as possible.

More precisely, the rotated and translated Gray codes produce $d2^d$ timewise edge disjoint Hamiltonian paths through the hypercube, d of them starting at each node. A *Hamiltonian path* in a graph visits all the nodes in the graph only once. *Timewise edge disjoint* means that, although the paths may share edges of the hypercube, no two paths traverse the same edge in the same direction on the same communication step. Consequently, assuming bidirectionality, there is no contention for communication channels. The data is circulated on these paths, but not retained, avoiding the memory waste. Furthermore, from a programming point of view, this approach is very simple and one can take advantage of the symmetry of the hypercube. The combination of the simplicity and the optimal use of communication bandwidth makes this communication pattern one of the fastest that a hypercube architecture can perform.

4.2. Complexity analysis. The complexity analysis of the algorithm is straightforward. The computation time, T_{arith}, scales like $N^2/2^d$, where N is the total number of bodies and d is the dimension of the hypercube. The communication time, T_{comm}, scales like $\lceil \frac{N}{2^d d} \rceil (2^d - 1)$; where $\lceil x \rceil$ denotes the ceiling of x, i.e., the smallest integer greater than or equal to x. Since there are $N/2^d$ bodies per node and d wires leaving each node, the first factor in T_{comm} represents the number of communication operations needed to empty a node of data. The second factor is the length of the path through the hypercube that the data must traverse.

Given a value of N such that $N/2^d \gg d$, T_{comm} is at first a linearly decreasing function of d. Then, as the number of dimensions increases, the number of required communication operations will eventually decrease to one and the second factor in the expression of T_{comm} starts to dominate. Because at least one such communication operation must be performed, even if there are less than d data items at a node, we get, for a fixed value of N,

$$\lim_{d \to \infty} T_{comm}(d) \simeq 2^d.$$

The function T_{comm} therefore exhibits a minimum that roughly occurs when the number of bodies per node is equal to the size of the bandwidth d. As the number of bodies per node decreases, one is making less and less effective use of the communications bandwidth of the hypercube and T_{comm} becomes a more significant portion of the total execution time. In fact, when $N/2^d \leq 1$, i.e., the number of bodies is much smaller than the number of processors, then the replicated orrery approach becomes more cost effective. Note that in the boundary case $N = P$, the total computational complexity reduces to $O(N)$ which agrees with the orrery algorithm.

4.3. Rotated and translated Gray codes. A d-dimensional hypercube is a graph with 2^d nodes labeled by the d-bit binary representation of the integers 0 to $2^d - 1$. Each bit in the d-bit representation is associated with a different dimension of the cube. There is an edge between two nodes i and j in the hypercube if and only if their binary representations differ in only one bit. We can think of the edges as traversing different dimensions of the hypercube.

A *Gray code* is a circuit of all binary d-tuples, such that only one coordinate position changes at each step. Thus, a Gray code represents a Hamiltonian path on the hypercube. There are many ways to construct Gray codes, but the most famous is the binary reflected Gray code [28]. To simplify the terminology, we will refer to this code as *the* Gray code. We define the *transition sequence* [28] corresponding to a Gray code as the list of positions that change at each step of the code or the list of dimensions traversed on the hypercube at each step of the circuit. Given the Gray code beginning at 0, we can "translate" it to node i by taking the *exclusive or* of i with the d-tuples in the original code. The transition sequence of the translated Gray code is identical to the original. In Fig. 9, we show the binary reflected Gray code for $d = 3$, the corresponding transition sequence, the associated Hamiltonian path on the cube, and the translated code starting at node 5 = 101.

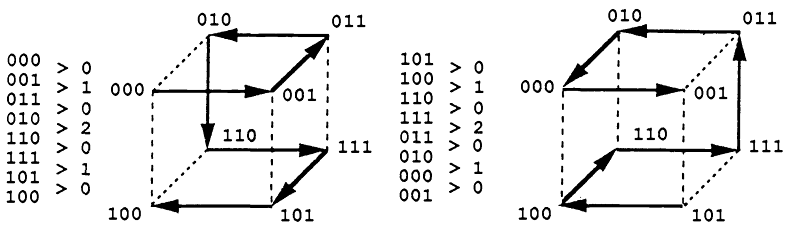


FIG. 9. *On the left is the binary reflected Gray code, the corresponding transition sequence, and the associated Hamiltonian path originating at node* 0 *on a three-dimensional cube. On the right, the same transition sequence is used to generate a translated code to node* 101 = 5 *and the associated Hamiltonian path.*

Given the Gray code beginning at 0, we can "rotate" it by performing circular shifts of the bits in the original code. In the tables below, we list the three rotated Gray codes and the corresponding transition sequences when $d = 3$. Note that generally, the transition sequence of the rotated code i is obtained by adding i modulo d to the transition sequence of the binary reflected Gray code.

Rotated Gray Codes								
Code 0	000	001	011	010	110	111	101	100
Code 1	000	010	110	100	101	111	011	001
Code 2	000	100	101	001	011	111	110	010

Transition Sequences							
Code 0	0	1	0	2	0	1	0
Code 1	1	2	1	0	1	2	1
Code 2	2	0	2	1	2	0	2

By taking all of the 2^d possible translations of the d rotated Gray codes beginning at 0, we can generate a set H of $d2^d$ Hamiltonian paths on the hypercube. This set H is more elegantly described as the paths generated by applying a group of $d2^d$ symmetries (rotations and translations) of the hypercube to the binary reflected Gray code[1] starting from node 0. These symmetries are the subgroup of all the hypercube symmetries that rotate the dimension numbers modulo d. It is straightforward to verify that two distinct symmetries of a hypercube transform a directed edge into two distinct directed edges. It follows that the collection of paths H have the nice property of being timewise edge disjoint: i.e., if $j \in 1, 2, \ldots, 2^d - 1$, then there is no overlap among the jth edges of the paths in H. This implies that if we have d packets of information in each node of the hypercube, we can circulate the data, one packet to a Hamiltonian path, in $2^d - 1$ communication steps. Furthermore, we can perform the arithmetic on the data in between each communication step eliminating the need for data storage.

[1]The use of the binary reflected Gray code is merely for convenience of implementation; any other Hamiltonian path will suffice.

5. CM-2 implementations.

5.1. The Connection Machine architecture. In this section we describe the basic design of the Connection Machine CM-2 so as to draw the distinction between the *slicewise* and *fieldwise* programming models. The Connection Machine CM-2 is composed of a microsequencer and a maximum of 65,536 single-bit processing elements. The processors run in SIMD (single instruction multiple data) mode, with the instruction stream broadcast by the sequencer. The sequencer is controlled by an external front-end machine, usually a SUN®.[2]

Each processor can have up to 1 Megabit of local RAM, with a single high-speed floating point unit for every 32 processors. There are 16 processors on a CM-2 chip and the chips are connected in a boolean d-cube topology, e.g., a 12-cube for a 64K processor machine.

The CM-2 supports three basic communication mechanisms. There is general pointer-based communication, known as the router, by which data can be exchanged between the memories of different processors. We refer to this type of communication as a *send*. The other two mechanisms involve direct control of the hypercube network by the user known as *cube swaps* and NEWS. These are used for more structured communication patterns. For example, the machine can be efficiently configured as a k-dimensional grid, which is automatically superimposed by the system software onto the boolean cube using a multidimensional Gray code mapping. Motion of the data from all processors to their nearest grid neighbors is known as a NEWS communication. These communications patterns are periodic in each dimension of the grid. Particularly useful primitives available on the Connection Machine computer are *scans*, parallel prefix operations [10] that combine computations and communications. In logarithmic time, these operators allow one to *spread* data through the CM-2, as well as accumulate summands (*plus-scan*) from each processor.

There are two programming models for the CM-2. In the *fieldwise* model, the storage of a 32-bit word would be allocated in 32 consecutive bits of a physical processor's memory. However, when performing floating-point computations, better performance is usually obtained by viewing the processors in a *slicewise* configuration. That is, we consider processing nodes on the machine to be the ensemble of a floating-point unit and the memory of the 32 associated physical processors of the CM-2. In this approach, a word is stored in 32-bit slices across the memories of the 32 processors in the node, i.e., 1-bit per processor. From this viewpoint, a 64K processor CM-2 becomes 2048 floating-point nodes, connected as an 11-dimensional hypercube with two bidirectional communication channels between connected nodes instead of one. In particular, this means that the communication complexity bound for the multiwire algorithm becomes

$$T_{comm} \approx \left\lceil \frac{N}{2^d 2d} \right\rceil (2^d - 1),$$

since $2d$ wires leave each node. Moreover, this model of the machine meshes efficiently with the way in which the floating-point units actually access data from their associated processors, i.e., in one cycle a 32-bit slice across processors is read into the floating-point unit. We should note that it is possible, by transposing data, to go back and forth between the slicewise and fieldwise models of the machine.

[2]SUN is a trademark of Sun Microsystems, Inc.

5.2. Implementation of the replicated orrery. Using the fieldwise programming model for the CM-2, we typically have $N \leq P \leq N^2$, thus the replicated orrery N-body algorithm is most appropriate. We describe the details of the implementation here.

In configuring the CM-2 as k-dimensional grids, we are required to restrict the lengths in each dimension to a power of 2. Thus, the data mapping of the replicated orrery onto a three-dimensional grid proceeds as follows. Pick n such that 2^n is the first power of 2 greater than or equal to N. Then we will actually use an orrery of size 2^n with only N valid entries. We are assuming that $N \leq P \leq N^2$, so we can write $P = 2^{n+m}$, where $m \leq n$. We will want to replicate the orrery $M = 2^m$ times. We configure a 2^{n+m} processor CM-2 as an 2^{n-m} by 2^m by 2^m cube (N/M by M by M). This configuration allows us to use versions of the parallel prefix scan operations to spread or accumulate data along any of the three coordinate directions in logarithmic time, and ensures the correct wraparound for grid communications. The cube mapping, complete with sends, spreads, and accumulates, is drawn in Fig. 7.

There will be two copies of the data for each body; one dynamic and one static. The static data is initially resident on the front face of the cube. It is sent to the top of the cube by a low density send. The data on the front face is spread back along the y-dimension of the cube, and the data on top is spread down along the z-dimension of the cube. The data is now distributed through the cube so that each plane parallel to the xz-axis contains a copy of the orrery. The computation now proceeds by computing a pairwise interaction according to the relevant force law and doing a one-dimensional wraparound NEWS communication along the x-axis. This process is repeated 2^{n-m} times until all N^2 pairwise interactions have been completed. Now the contributing forces for each individual body are accumulated by doing a plus-scan along the y-axis back to the front face of the cube.

Because of the power of 2 restrictions noted above, the time required to do the computation for any value of N between 2^n and $2^{n+1} - 1$ is the same. As N increases above the next power of 2, we can reconfigure the grid to the appropriate size.

5.3. Implementation of the multiwire algorithm. Using the slicewise model of the CM-2, we are generally in the case where $P < N$, and so we use the multiwire algorithm described above.

If we have d words at every node, and if we wish to circulate the data using the Hamiltonian paths described in the §3, one might expect that we would either need to attach some labeling information to be transmitted with the data or else use a look-up table that specifies the dimension over which data needs to be sent. Neither of these operations is without cost. In our implementation, as we shall now describe, we avoid both of these types of methods. In fact, the information in the transition sequence for the binary reflected Gray code is all that is needed.

Using the slicewise model, the CM-2 architecture allows for the loading of as many as $2d$ words to be sent over the d dimensions simultaneously, two words in each dimension. Typically, one loads one set of d words for the d dimensions followed by another set of d words. For simplicity, we will concentrate on one set with the understanding that the other set follows exactly the same pattern. Another important feature of the CM-2 architecture is that it is particularly efficient to load consecutive words, or more generally, words with a constant stride, to be sent over consecutive dimensions.

To illustrate how the CM-2 works, let us first assume that we only have one word in each processor and each of these words will follow the Hamiltonian path specified by the transition sequence of the Gray code: $g = \{0, 1, 0, 2, 0, 1, \ldots\}$. Furthermore, assume we have only one memory location (location 0) in each node where the word is stored.

Then Algorithm 1 is simply:

```
For $k = 1, 2, \ldots, 2^d - 1$
In each node, the data at location 0 is
1. Loaded to be sent over dimension $g_k$.
2. Transmitted.
3. Stored in location 0.
```

Now, if we have d words in each node at memory locations 0 through $d-1$, we can quite readily ensure that the word at location j in each node follows the jth transition sequence. The method is to store this word at location j as it passes through each node like a traveler who prefers the same hotel in each city. Then, at step k, we send the data in each node at location j over dimension $g_k + j \pmod{d}$. To take advantage of the Connection Machine's ability to load consecutive words, we in fact use Algorithm 2 below.

```
For $k = 1, 2, \ldots, 2^d - 1$
In each node
1. Data at locations 0 through $d - g_k - 1$ is loaded to
   be sent over dimensions $g_k$ through $d - 1$, respectively.
2. Data at location $d - g_k$ through $d - 1$ is loaded to
   be sent over dimensions 0 through $g_k - 1$, respectively.
3. Data is transmitted.
4. Data received over dimensions $g_k$ through $d - 1$ is
   stored at locations 0 through $d - g_k - 1$, respectively.
5. Data received over dimensions 0 through $g_k - 1$ is
   stored at locations $d - g_k$ through $d - 1$, respectively.
```

Note that steps 2 and 5 are vacuous if $g_k = 0$, which is half of the time. The actual code to perform this operation is not much longer than the pseudocode above.

In the real algorithm, we assume there is a multiple of $2d$ bodies in each node, and each body may contain more than one word of information. In the example described in the next section, there are two words of coordinate information and also a coefficient, giving three words for each body. Since we can transmit $2d$ words simultaneously, we loop over all the data until it is all transmitted. After the data is transmitted, we perform a computation such as the one described in the next section, and then we repeat for $k = 1, 2, \ldots, 2^d - 1$.

The ideas we presented in this section have now been coded up as a Connection Machine Scientific Software Library Primitive. See [12] for details.

6. A sample application.

6.1. Description. As an application example, we computed the velocities of a collection of interacting point vortices in two dimensions. This N-body problem is at the heart of the vortex method to solve the Navier–Stokes equation for incompressible viscous flows [30]. A Lagrangian perspective of the vorticity-stream function formulation of the Navier–Stokes equation leads to the introduction of discrete vortex elements that interact according to a Biot–Savart-type force law. To avoid potential numerical instability caused by very close encounters of particles, finite-size vortex elements should be used. However, in these timing runs, we have considered the interaction of point vortices.

In two dimensions, the (u, v) velocity components of a point vortex at position (x, y) are given by the following formulae:

$$u(i) = \frac{1}{2\pi}\sum_{j=1}^{N}\frac{c(j)\Delta y(i,j)}{\Delta r(i,j)^2}, \qquad i = 1,\ldots,N,$$

$$v(i) = \frac{-1}{2\pi}\sum_{j=1}^{N}\frac{c(j)\Delta x(i,j)}{\Delta r(i,j)^2}, \qquad i = 1,\ldots,N,$$

with

$$\Delta x(i,j) = x(j) - x(i), \qquad \Delta y(i,j) = y(j) - y(i),$$

$$\Delta r(i,j)^2 = \Delta x(i,j)^2 + \Delta y(i,j)^2, \quad \text{and} \quad c(j) = \text{ circulation of point vortex } j.$$

6.2. Timings and analysis. We performed several calculations for both the replicated orrery and the multiwire algorithm with a varying number of particles on various machine sizes. Timing results, in seconds, are shown in the Tables 1 and 2 below. Figure 10 is a graph of the performance given in those tables for the two algorithms on an 8K processor CM-2. Both time and N, the number of bodies, are graphed using a logarithmic scale. We have made the piecewise constant performance behavior of the replicated orrery algorithm clearly visible. Timings for the replicated orrery algorithm were run on 4K and 8K processor configurations. The multiwire timings were obtained on CM-2 configurations with 2^{d+5} processors for $d = 4, 5, 7, 8, 9$, i.e., with 2^d floating point nodes. Typically, 14,000 vortices interact in 2 seconds on a 512 floating-point node machine (16K processors) using the multiwire algorithm, and 17 seconds on an 8K processor machine using the replicated orrery. Counting the divide operation as one operation, the execution rate of the multiwire algorithm is about 5.2 Gflops on a fully configured CM-2 (2048 nodes and 64K processors).[3]

It is difficult to compare timings of these two algorithms because they use different models of the machine (slicewise vs. fieldwise). For a crude approximation, on a fixed CM-2 configuration, one can compare the timing for N bodies using the multiwire algorithm to the timing for the next largest power of 2 bodies using the replicated orrery. Doing this, for an equivalent number of bodies, the multiwire algorithm outperforms the replicated orrery by a factor of 3 to 5. Some loss in performance of the replicated orrery algorithm is due to the necessity of transposing fieldwise data back and forth for the floating-point units. There is an additional loss due to the fact that the CM-2 fieldwise model forces the sizes in the replicated orrery to be powers of 2. Thus, one does as much arithmetic as if N were rounded up to the next power of 2.

Keeping the number of bodies constant, the time to compute the velocity components appears inversely proportional to the number of floating-point nodes of the hypercube. This is the kind of behavior we expect given the complexity analysis of both algorithms. Note that, for a fixed number of processors, the proportion of time spent in the communications for the replicated orrery decreases very slowly as the number of

[3]On the 32-bit floating-point unit of the CM-2, a divide operation requires the explicit implementation of two Newton–Raphson iterations of the initial value which is provided by an internal look-up table. This amounts to a total of six atomic operations that we count as one.

TABLE 1

Replicated orrery			
Processors	Bodies	T_{total}	T_{comm}
4K	1024	.15	40%
	2048	.6	30%
	4096	2.5	30%
	8192	8.5	24%
	16384	30	20%
8K	1024	.08	44%
	2048	.3	30%
	4096	1.3	28%
	8192	5.1	29%
	16384	16.7	24%

TABLE 2

Multiwire Algorithm			
Processors	Bodies	T_{total}	T_{comm}
512 ($d = 4$)	1792	1.0	4%
	3584	4.0	2%
1K ($d = 5$)	1792	.52	7%
	3584	2.0	4%
	7168	7.8	2%
	14336	30.8	1%
4K ($d = 7$)	1792	.16	15%
	3584	.54	9%
	7168	2.0	5%
	14336	7.8	3%
8K ($d = 8$)	1792	.11	28%
	3584	.3	10%
	7168	1.07	6%
	14336	4.04	3%
16K ($d = 9$)	3584	.2	50%
	7168	.6	16%
	14336	2.1	9%

bodies increases. In the multiwire algorithm the proportion of communication time is much smaller and the decrease is more rapid. We get a more accurate picture of what is going on in the multiwire algorithm from Fig. 11 which plots the actual speedup of the implementation against the optimal speedup of 2^d (the number of processors). We calculated the speedup by taking the ratio of T_{arith} times the number of processors and the total execution time of the calculation. Note that again the curves behave as predicted by the complexity analysis. For a fixed hypercube dimension, as the number of bodies increases the speedup approaches optimality. In particular, over the range of test cases we ran, T_{comm} varies from a few percent to fifty percent of the total execution time as the number of bodies approaches the number of processors.

Acknowledgments. We would like to thank Washington Taylor for his help in the first implementation of the replicated orrery and Adam Greenberg for running a set of timings of the replicated orrery for us.

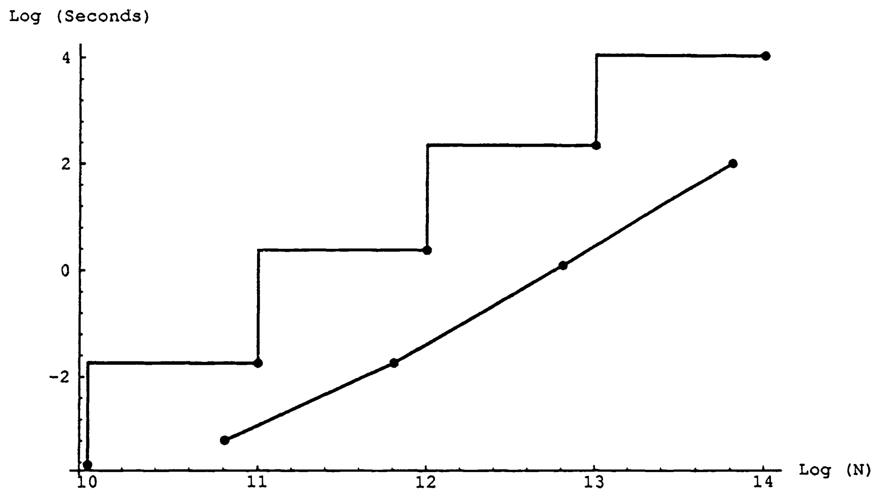


FIG. 10. *Comparison of replicated orrery and multiwire orrery timings on an* 8K*processor* CM-2. *Both time and the number of bodies N are graphed on a log scale.*

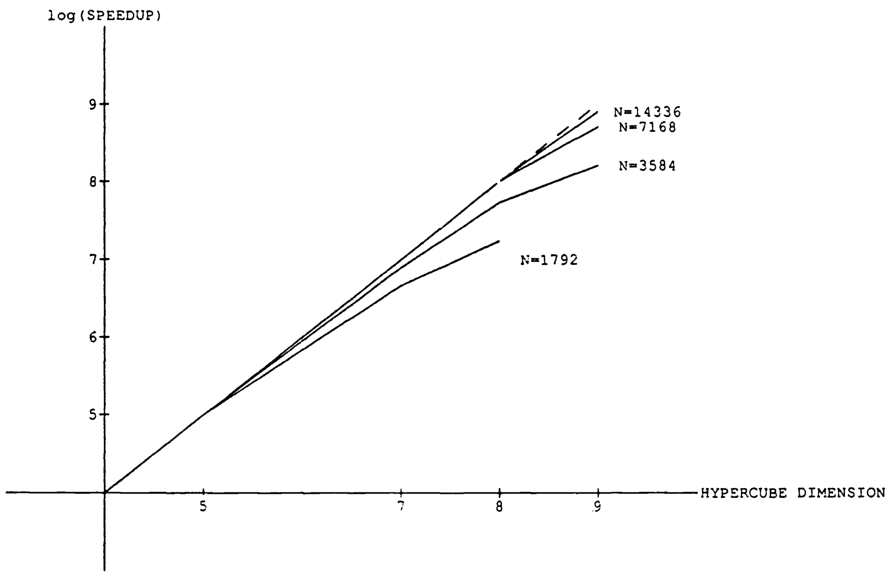


FIG. 11. *Speedup on the* CM-2 *for the direct interaction of N bodies in two dimensions, with* N =1792, 3584, 7168, 14,336.

REFERENCES

[1] C. R. ANDERSON, *A method of local corrections for computing the velocity field due to a distribution of vortex blobs*, SIAM J. Numer. Anal., 22 (1986), pp. 413–440.

[2] A. W. APPEL, *An efficient program for many-body simulation*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 85–103.

[3] J. F. Applegate, M. R. Douglas, Y. Gursel, P. Hunter, C. Seitz, and G. J. Sussman, *A digital orrery*, IEEE Trans. Comput., 84 (1985), p. 822.

[4] S. B. Baden and E. G. Puckett, *A fast vortex code for computing* 2D *viscous flow*, J. Comput. Physics, 91 (1990), pp. 278–297.

[5] J. Barnes, *Evolution of compact groups and the formation of elliptical galaxies*, Nature, 338 (1989), pp. 123–126.

[6] ———, *Multiple-timestep* N*-body algorithms for parallel computers*, Institute for Advanced Study, Princeton University, Princeton, NJ, 1987, preprint.

[7] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[8] D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis, *Optimal communication algorithms for hypercubes*, J. Parallel Distributed Comput., 11 (1991), pp. 263–275.

[9] A. Beck, M. N. Bleicher, and D. W. Crowe, *Excursions into Mathematics*, Worth Publisher, Inc., New York, 1976.

[10] G. E. Blelloch, *Scans as primitive parallel operations*, Proc. Internat. Conf. on Parallel Processing, 1987.

[11] J. P. Brunet, A. Edelman, and J. P. Mesirov, *Proceedings Supercomputing* '90, IEEE Computer Society Press, Los Alamitos, CA, pp. 748–752.

[12] J. P. Brunet and S. L. Johnsson, *All-to-all broadcast and applications on the Connection Machine*, Tech. Rep., TMC-189/BA, Thinking Machines Corp., Cambridge, MA.

[13] J. Dénes and A. D. Keedwell, *Latin Squares and Their Applications*, Academic Press, New York, 1974.

[14] G. C. Fox, P. Hipes, and J. Salmon, *Practical parallel supercomputing: examples from chemistry and physics*, Proceedings Supercomputing '89, ACM Press, Reno, NV, 1989, pp. 58–70.

[15] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1986), pp. 325–348.

[16] A. F. Ghoniem, A. J. Chorin, and A. K. Oppenheim, *Numerical modeling of turbulent flow in a combustion tunnel*, Philos. Trans. Roy. Soc. London A., 304 (1982), pp. 303–325.

[17] H. Heller, H. Grubmüller, and K. Schulten, *Molecular dynamics simulation on a parallel computer*, Molec. Simul., 5 (1990), pp. 133–165.

[18] L. Hernquist, *Tidal triggering of starbursts and nuclear activity in galaxies*, Nature, 340 (1989), pp. 687–691.

[19] W. D. Hillis and J. Barnes, *Programming a highly parallel computer*, Nature, 326 (1987), pp. 27–30.

[20] C. T. Ho, personal communication, 1991.

[21] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981.

[22] S. L. Johnsson, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel and Distributed Comput., 4 (1987), pp. 133–172.

[23] S. L. Johnsson and C. T. Ho, *Optimum broadcasting and personalized communication in hypercubes*, IEEE Transactions on Computers, 38 (1989), pp. 1249–1268.

[24] J. Katzenelson, *Computational structure of the N-body problem*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 787–815.

[25] A. Leonard, *Computing three-dimensional incompressible flows with vortex elements*, Ann. Rev. Fluid Mech., 17 (1985), pp. 523–559.

[26] J. P. Mesirov and W. Taylor, *A Family of* N*-body Solvers*, 1988, manuscript.

[27] A. R. C. Raine, D. Fincham, and W. Smith, *Systolic loop methods for molecular dynamics simulation using multiple transputers*, Comput. Phys. Comm., 55 (1989), pp. 13–30.

[28] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[29] H.-W. Rix and S. D. M. White, *The dynamic of dumb-bell galaxies*, Monthly Notices Roy. Astronom. Soc., 240 (1989), pp. 941–656.

[30] J. A. Sethian, J. P. Brunet, A. Greenberg, and J. P. Mesirov, *Two-dimensional, viscous, incompressible flow on a massively parallel processor*, J. Comput. Phys., 1989, 101 (1992), pp. 185–206.

[31] J. A. Sethian and K. Gustafson, *A Brief Overview of Vortex Methods, in Vortex Methods and Vortex Motion*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991.

[32] W. Smith, *Molecular dynamics on hypercube parallel computers*, Comput. Phys. Comm., 62 (1991), pp. 229–248.

[33] A. Windemuth and K. Schulten, *Molecular dynamics simulation on the Connection Machine*, Molec. Simu., 5 (1991), pp. 353–361.

[34] F. Zhao and S. L. Johnsson, *The parallel multipole method on the Connection Machine*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1420–1437.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1159–1173, September 1993

ANALYSIS OF THE MULTIGRID FMV CYCLE ON LARGE-SCALE PARALLEL MACHINES*

RAY S. TUMINARO† AND DAVID E. WOMBLE†

Abstract. On serial computers it is well known that the multigrid FMV cycle is preferable to the V cycle both asymptotically and in practical use over a wide range of applications. However, on massively parallel machines, the parallel efficiency of the FMV (full multigrid V cycle) scheme is noticeably lower than that of the V cycle due to a large percentage of time spent on coarse grids. Thus the question arises: are the additional coarse grid computations within the FMV cycle warranted on massively parallel machines? To answer this, a number of issues are addressed regarding parallel FMV cycles: what efficiencies can be achieved; how do these compare with V cycle efficiencies; are FMV cycles still preferable to V cycles in a massively parallel environment?

A model is used to analyze the efficiency of both FMV and V cycles as a function of relaxation efficiency. Using this model, the standard FMV grid-switching criterion is modified to incorporate the efficiency of the coarse grid processing. Numerical results obtained from a multigrid implementation on a 1024-processor nCUBE 2 are used in conjunction with the model to quantify the performance and efficiency of the FMV cycle. Finally, comments are made regarding limitations of parallel processors based on FMV efficiencies.

Key words. parallel computing, multigrid, FMV cycle

AMS subject classifications. 65Y05, 65M55

1. Introduction. Multigrid algorithms are used within a variety of scientific disciplines and are among the fastest iterative methods for many problems of practical interest. Unlike conventional schemes, multigrid methods use a hierarchy of coarse meshes to accelerate convergence on the finest mesh. Depending on the order and frequency of these coarse grid computations, different multigrid variations are possible (e.g., V cycle, FMV cycle, W cycle, F cycle). While tradeoffs between these variations have been studied on serial computers, far less is known on parallel machines. For the most part, only the parallel V cycle has been studied extensively (see, [2], [4]–[6], and [8]–[13]). Given that the FMV cycle (full multigrid V cycle) is usually faster than the V cycle on serial computers, we feel it is appropriate to contrast the parallel performance of the FMV and V cycle iterations.

The key tradeoff between V and FMV cycles lies in the choice of initial guess. Specifically, the standard V cycle begins the computation on the finest grid using an arbitrary initial guess. The FMV cycle, on the other hand, starts with the coarsest grid, which is used to compute a good initial approximation on the finest grid. Thus, a greater percentage of time is spent on coarse grid computations (and hence less time on the fine grid) within the FMV iteration. Since coarse grid processing is inexpensive on serial computers, the FMV approach is usually very beneficial. However, on massively parallel computers the situation can be quite different. In particular, it is much more difficult to efficiently parallelize coarse grid computations due to the fact that there is less parallelizable work. Consequently, the cost differential between fine and coarse grid processing may be so small that the overall execution time of a simple FMV cycle is not necessarily better than that of the V cycle.

* Received by the editors December 30, 1991; accepted for publication (in revised form) November 8, 1992. This work was supported by the Applied Mathematical Science Program, U.S. Department of Energy, Office of Energy Research and was performed at Sandia National Laboratories operated for the U.S. Department of Energy contract DE-AC04-76DP00789.

† Applied and Numerical Mathematics Division, Sandia National Laboratories, Albuquerque, New Mexico 87185.

In this paper, we analyze the FMV cycle on massively parallel computers. The purpose of this analysis is to quantify the performance and efficiency of an FMV iteration as a function of the relaxation efficiency. That is, how much efficiency (from a parallel computing standpoint) is lost due to the heavy use of coarse grids in the FMV cycle? To answer this, we review the grid switching criterion within FMV cycles. This criterion determines when the coarse grid approximation is interpolated for use as an initial guess on the next finest grid. The switching criterion is a function of computation times for fine and coarse grid iterations. Since these times depend on parallel efficiency, we modify the switching criterion to incorporate parallel efficiency. A model of the FMV cycle as a function of relaxation operations is developed. We compare the V and FMV cycles using this model and numerical results obtained from a multigrid implementation on a 1024-processor nCUBE 2. Based on this analysis, conclusions are drawn about the merits of the parallel FMV cycle as well as the limitations of parallel processors.

2. The FMV cycle. We begin with a brief description of the V and FMV cycles. It is assumed that the reader is already familiar with the multigrid method (see [3] for introductory material).

Consider the discrete problems

$$A_i u_i = f_i,$$

where A_i is an $n \times n$ matrix corresponding to the discretization of a partial differential equation (PDE) problem on grid $\mathcal{G}_i$ with mesh spacing $h = 2^{-i}$, and f_i and u_i are n-vectors. One iteration of a multigrid V cycle can be written as follows:

```
ALGORITHM 1. $u_i \leftarrow \mathrm{MV}(u_i, f_i)$.
    1. Relax $\nu_1$ times on $A_i u_i = f_i$ with initial guess $u_i$.
    2. If $i > 0$,  $f_{i-1} \leftarrow I_i^{i-1}(f_i - A_i u_i)$    /* $I_i^{i-1}$: projection from $\mathcal{G}_i$ to $\mathcal{G}_{i-1}$ */
                $u_{i-1} \leftarrow 0$
                $u_{i-1} \leftarrow \mathrm{MV}(u_{i-1}, f_{i-1})$
                $u_i \leftarrow u_i + I_{i-1}^i u_{i-1}$    /* $I_{i-1}^i$: interpolation from $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$ */
       Endif
    3. Relax $\nu_2$ times on $A_i u_i = f_i$ with initial guess $u_i$.
```

The V cycle procedure starts with an initial guess and uses relaxation iterations on a series of coarser and coarser grids to improve the guess. To enhance the V cycle, an initial approximation can be obtained by first "solving" the PDE problem on a coarser grid using the multigrid procedure. This leads to the FMV cycle written recursively as follows:

```
ALGORITHM 2. $u_i \leftarrow \mathrm{FMV}(f_i)$.
    1. If $i > 0$,  $f_{i-1} \leftarrow I_i^{i-1} f_i$
                $u_{i-1} \leftarrow \mathrm{FMV}(f_{i-1})$
                $u_i \leftarrow I_{i-1}^i u_{i-1}$
       Endif
    2. $u_i \leftarrow \mathrm{MV}(u_i, f_i)$   $\eta_i$ times,
```

where the parameter η_i determines how many V cycles are performed on level i.

On serial machines, the FMV cycle is faster than the V cycle both asymptotically and in practical use. In particular, it is possible to show that the FMV cycle requires $O(N)$ operations on a serial computer compared to $O(N \log N)$ operations for the V cycle where N is the number of grid points on the fine grid.[1] However, on parallel machines the relative performance of the two schemes is not as clear. In particular, multigrid methods are parallelized by performing computations within a grid in parallel but still processing each grid in the hierarchy in sequence. Therefore, the parallel efficiency on each individual grid needs to be considered. Unfortunately, when computing on coarse grids, many processors are inactive due to the lack of computational work. Consequently, the FMV iteration (which uses coarse grids heavily) makes significantly less efficient use of parallel processors than the V cycle. Thus, even though the FMV cycle is superior on serial machines, parallel V and FMV cycles require the same number of operations asymptotically, $O(\log^2 N)$, on a parallel machine with N processors. The remainder of this paper explores the relationship between the two cycling strategies on parallel machines.

3. Switching criteria. We begin our discussion of the parallel FMV process by considering the proper choice of the η_i's. Specifically, we wish to minimize the overall run time in the FMV process. To do this, we balance the time saved on a fine grid (due to a better initial guess) versus the time required to produce an approximation on the coarser grid. Thus, η_i depends on the run times required for V cycles at levels i and $i-1$. For a serial computer, the ratio of these times is fixed. However, for a parallel computer, it is necessary to consider the efficiency of the computer which, in turn, depends on machine parameters and grid size. The basic idea is given in [1] for serial computations. Here, we generalize the arguments to include parallel processors.

To obtain a grid switching criterion, it is useful to first develop an expression relating the errors on different meshes. In particular,

$$u(\mathcal{G}_i) = \tilde{u}_i + e_i + E_i, \tag{1}$$

where

$$\begin{aligned} u(\mathcal{G}_i) &= \text{exact PDE solution evaluated on } \mathcal{G}_i, \\ u_i &= \text{exact discrete solution on grid } \mathcal{G}_i, \\ \tilde{u}_i &= \text{current numerical approximation on } \mathcal{G}_i, \\ E_i &= u(\mathcal{G}_i) - u_i \qquad \text{(truncation error)}, \\ e_i &= u_i - \tilde{u}_i \qquad \text{(algebraic error)}. \end{aligned}$$

If we assume that the approximation on $\mathcal{G}_{i-1}$ is interpolated such that the interpolation errors are negligible (i.e., sufficiently high order), a simple expression relating algebraic and truncation errors can be derived. Specifically, we use (1) on two adjacent grids to obtain

$$e_i^0 - I_{i-1}^i e_{i-1}^* = I_{i-1}^i E_{i-1} - E_i, \tag{2}$$

where 0 denotes an initial approximation obtained by interpolating the current solution from the previous grid, $*$ denotes the final approximation which is to be interpolated to the next grid, and I_{i-1}^i denotes a suitably high-order interpolation operator from $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$. In the remainder of this section we omit the interpolation operator. However, it is

[1]This operation count corresponds to obtaining a solution whose accuracy is approximately the same magnitude as the discretization error. It is assumed that the V cycle convergence rate is independent of the mesh spacing. This is true when multigrid methods are applied to many elliptic problems.

understood that interpolation takes place whenever two quantities defined on different grids are combined.

Since the goal is to minimize work, we switch from $\mathcal{G}_{i-1}$ to $\mathcal{G}_i$ when the greater error reduction associated with a V cycle on $\mathcal{G}_i$ outweighs the increased cost of operating on $\mathcal{G}_i$ (see the Appendix for details). This occurs when

$$\|e_i^0\| = \xi_{i,p}\ \|e_{i-1}^*\|, \tag{3}$$

where

$$\xi_{i,p} = \frac{\tau_{i,p}(V)}{\tau_{i-1,p}(V)}$$

and $\tau_{i,p}(V)$ is the V cycle execution time starting on $\mathcal{G}_i$ using p processors. Combining (2) and (3) and assuming that the local truncation errors satisfy

$$E_i \approx 2^{-q} E_{i-1} \tag{4}$$

for a qth-order discretization of a PDE, we obtain

$$\|e_{i-1}^*\| \le \left(\frac{1-2^{-q}}{\xi_{i,p}-1}\right) \|E_{i-1}\|. \tag{5}$$

This expression can be used for determining convergence. For now, we use (5) to obtain the required algebraic error reduction on $\mathcal{G}_{i-1}$ so that (3) is satisfied. In particular, (3), (4), and (5) on $\mathcal{G}_{i-2}$ yield

$$\|e_{i-1}^0\| \le \xi_{i-1,p} \left(\frac{1-2^{-q}}{\xi_{i-1,p}-1}\right) 2^q \|E_{i-1}\|. \tag{6}$$

If, in (5) and (6) we assume equality,[2] the necessary error reduction condition follows

$$\begin{aligned} \frac{\|e_{i-1}^*\|}{\|e_{i-1}^0\|} &\approx \left(\frac{\xi_{i-1,p}-1}{\xi_{i,p}-1}\right) \frac{2^{-q}}{\xi_{i-1,p}} \\ &\approx \frac{\tau_{i-1,p}(S)}{\tau_{i,p}(S)} 2^{-q} \\ &\approx \frac{\varepsilon_{i,p}(S)}{\varepsilon_{i-1,p}(S)} 2^{-q-d}, \end{aligned} \tag{7}$$

where

$$\tau_{i,p}(S) = \tau_{i,p}(V) - \tau_{i-1,p}(V)$$

is the time for operations associated with $\mathcal{G}_i$ and

$$\varepsilon_{i,p}(S) = \frac{\tau_{i,1}(S)}{p\tau_{i,p}(S)}$$

is the efficiency of the operations corresponding to $\tau_{i,p}(S)$. Since the ratio of the smoother times is representative of the ratio between operations on $\mathcal{G}_i$ and $\mathcal{G}_{i-1}$, we

[2]We must assume equality to account for the worst-case situation and guarantee that the final error is sufficiently small.

take $\tau_{i,p}(S)$ to be the time required for smoothing on $\mathcal{G}_i$. Thus, the formula states that the switching of grids is determined by the ratio of the relaxation time on $\mathcal{G}_i$ and $\mathcal{G}_{i-1}$. No other operations need be considered. Note that in the serial computing case (i.e., $p = 1$), the ratio between the smoother times at two levels is usually 2^d (where d is the dimension of the problem) giving an error reduction criterion of 2^{-q-d} which is consistent with [1]. Furthermore, in the case of one point per processor and negligible communication costs, we have $\tau_{i-1,p}(S) \approx \tau_{i,p}(S)$ yielding a reduced switching criterion of 2^{-q}. Thus, the net effect of the new switching criterion is to shift more of the computation away from the coarser grids where the parallel efficiency is low to the finer grids where the parallel efficiency is higher.

4. Experimental results. A parallel multigrid algorithm was implemented on a 1024 processor nCUBE 2 system. On each level the algorithm uses one red-black Gauss–Seidel iteration as a smoother. Bilinear interpolation and full weighted restriction are used for grid transfer operators within the V cycle. Bicubic interpolation is used between grids in the FMV process (i.e., interpolation errors can be ignored in the analysis). Parallelization is obtained by decomposing the problem spatially in that each processor maintains a region of the domain. On coarse grids, it is possible that some processors may contain no points within their subdomain. In this case, processors without grid points perform redundant work in parallel to the useful work proceeding on other processors.

The algorithm is applied to a two-dimensional central difference approximation of a model anisotropic problem

$$\frac{\partial^2 u}{\partial x^2} + \gamma \frac{\partial^2 u}{\partial y^2} = f(x, y) \tag{8}$$

defined on the unit square with Dirichlet boundary conditions, $\gamma = .18$, and suitable right-hand side. Three cases are considered here.

• V cycle. $\lceil -\log_2 N/\log_2 \beta \rceil$ V cycle iterations are performed where β is the V cycle convergence rate. This corresponds to the number of iterations required to reduce errors to level of truncation error assuming that the initial guess on the fine grid satisfies: $\|e_i^0\| \approx 1$ and $\|E_i\| \approx h^2$, where h is the mesh spacing. β is .77 for our experiments.[3]

• FMVS cycle. An FMV cycle is performed using the standard grid switching criterion. That is, the error is reduced by a factor of 16 on the coarse grids. This corresponds to $\lceil -4/\log_2 \beta \rceil$ V cycles on each of the coarse grids before interpolation to the next grid. On the finest grid, (6) is used (with $p = 1$) to determine the number of iterations required to reduce the algebraic error below the level of truncation error. This corresponds to $\lceil -2/\log_2 \beta \rceil$ iterations on the finest grid. Note that the number of iterations is always increased to an integer value.

• FMVP cycle. An FMV is performed using the parallel grid switching criterion. The ratio of the smoother times in (7) is estimated using a timing model (see (12)). On the finest level (6) is used to determine the final number of iterations. For a 64×64 grid on 1024 processors, the following number of V cycles were performed on each level starting from the second coarsest grid and ending with the finest grid: 5, 6, 6, 4, and 10. This corresponds to less work on the coarse grid and more work on the fine grid when compared to the FMVS cycle which performs the following V cycles: 11, 11, 11, 11, and 6.

[3]Experiments were performed with a somewhat poor V cycle convergence rate so that the effects of rounding the grid switching criterion (e.g., when the switching criterion determines that 1.8 V cycles is optimal, we perform 2 V cycles) would be minimal. However, γ was not chosen so small as to make the V cycle with point smoothing a bad choice of algorithm.

In Tables 1 and 2, we compare the total run times and efficiencies[4] for the V, FMVS, and FMVP schemes on both a 64×64 grid and a 1024×1024 grid. For the larger problem, the relative performance is similar to the serial computing case. That is, both FMV schemes outperform the V cycle method by a substantial margin despite the fact that the overall FMV efficiencies are not good (even for the large problem). That is, the heavy use of coarse grids results in fairly low FMV efficiencies. On the smaller grid the situation is much worse. In particular, the time differential between the the V cycle and FMV cycles is not large and the overall efficiencies are extremely low. In other words, when efficiencies are low, it is not clear if the FMVS iteration is superior to the V cycle. In the remainder of this paper, the relationship between V and FMV cycles is explored in more detail.

TABLE 1
Execution times (in secs.) and efficiencies (shown in parentheses) on the 64×64 *problem.*

Cycle\Procs	1	4	16	64	256	1024
V	22.8 (1.0)	7.2 (.79)	2.7 (.53)	1.4 (.25)	1.1 (.08)	1.0 (.02)
FMVS	8.5 (1.0)	3.1 (.69)	1.5 (.35)	1.1 (.12)	0.98 (.03)	0.98 (.008)
FMVP	8.5 (1.0)	3.1 (.69)	1.4 (.38)	1.0 (.13)	.90 (.04)	.70 (.01)

TABLE 2
Execution times (in secs.) and efficiency (shown in parentheses) on the 1024×1024 *problem.*

Cycle\Procs	64	256	1024
V	151.8 (1.0)	42.2 (.90)	13.6 (.70)
FMVS	35.3 (1.0)	11.5 (.77)	5.2 (.42)
FMVP	35.3 (1.0)	11.1 (.80)	4.7 (.47)

5. Execution model. We can express the execution time of V and FMV cycles as a function of the time on each level. For the V cycle we have

$$\tau_{i,p}(V) = \sum_{j=0}^{i} \tau_{j,p}(S). \tag{9}$$

Similarly, the FMV cycle can be expressed as

$$\tau_{i,p}(F) = \sum_{j=0}^{i} \eta_j \tau_{j,p}(V). \tag{10}$$

Note that when $\eta_j = 1$, we have the same relation between FMV and V cycles as we have between the V cycle and the smoother. This will be discussed in more detail in the next section. Using the following definition of efficiency

$$\varepsilon = \frac{\text{serial time}}{p \ * \ \text{time using } p \text{ processors}},$$

we can obtain expressions relating the efficiency of the smoother, V cycle, and FMV cycle (similar to those in [13]). In particular, we have

[4]Efficiencies are calculated based on the execution time on the smallest number of processors used for the problem.

$$\varepsilon_{l,p}(V) = \frac{2^{d(l+1)} - 1}{(2^d - 1)\sum_{i=0}^{l} \frac{2^{di}}{\varepsilon_{i,p}(S)}},$$

$$\varepsilon_{l,p}(F) = \frac{\sum_{i=0}^{l} \eta_i \; 2^{di}}{\sum_{i=0}^{l} \frac{\eta_i \; 2^{di}}{\varepsilon_{i,p}(V)}},$$

and for $\eta_i = \eta_j$ and large l, we get

$$\varepsilon_{l,p}(F) \approx \frac{2^{d(l+1)} - 1}{(2^d - 1)^2 \;\sum_{i=0}^{l} \frac{(l-i+1)2^{d(i-1)}}{\varepsilon_{i,p}(S)}}.$$

Note that the overall efficiencies are just weighted harmonic averages of the efficiencies of the smoother. Since the larger weights are applied to the fine grid efficiencies, we can conclude that the efficiencies of the fine grid smoother dominate the overall efficiency. However, the fine grid weights for the FMV cycle are lower relative to the fine grid weights for the V cycle (due to the $l - i + 1$ term). Thus, we would expect that the FMV efficiencies are governed much more by coarser grids than the V cycle. In the next section, we use this model to explore the performance of V and FMV schemes in a variety of situations.

6. Performance of the V, FMVS, and FMVP cycles. A program was written using the model and grid switching criterion given in the previous sections to estimate run time and efficiencies under different circumstances. We use this model to explore the performance of the different multigrid cycles beginning with the special situation of one grid point per processor. In our model we assume that the time required on each grid level is constant (i.e., $\tau_{i,p}(V) = 1$). In this case, (9) and (10) can be used in conjunction with convergence and grid switching criteria to determine the required number of operations.[5] For example, one V cycle takes α operations where α is the number of grid levels and a total of $-2\alpha/\log_2(\beta)$ iterations to reduce the algebraic error below truncation error. A summary of the operations counts for the different cycles is given below.

V (2d & 3d):	$\gamma\alpha^2$,
FMVS (2d):	$\gamma\alpha^2$,
(3d):	$\gamma[\frac{5}{4}\alpha(\alpha-1) + \alpha\log_2(\sqrt{24/7})]$,
FMVP (2d & 3d):	$\gamma[\frac{1}{2}\alpha(\alpha-1) + \alpha\log_2(\sqrt{3\alpha})]$,

where

$$\alpha = log_2(N^{1/d}), \qquad \gamma = -2/\log_2(\beta), \tag{11}$$

β is the convergence rate of the V cycle, N is the total number of grid points, and d is the dimension of the problem. Note that asymptotically the FMVP cycle has the fastest execution time, which is less than a factor of two better than the FMVS execution time. Furthermore, the FMVS cycle is competitive with the V cycle even in the extreme case

[5]The number of iterations are based on the same assumptions given in §4. The only difference is that we use the V cycle run times to determine the η_i's in conjunction with (6). Additionally, we note that in this analysis the number of V cycles need not be an integer.

of one point per processor. In particular, the FMVS cycle requires the same number of operations as the V cycle on two-dimensional problems while only slightly more in the three-dimensional case. Thus, the FMV cycle (even without the new switching criterion) performs almost as well as the V cycle despite a large percentage of time spent on coarse grids.

Of course, in most situations we have more grid points than processors. To model this, we consider run times that are proportional to the number of grid points per processor. That is,

$$\tau_{i,p}(S) = C\left\lceil \frac{N}{2P} \right\rceil,$$

where N is the number of grid points on $\mathcal{G}_i$, P is the number of processors, and C is a constant independent of N and P. Note that the factor of two occurs because half the processors are idle during a red/black Gauss–Seidel iteration when each processor contains only one point. This model might correspond to the case where communication costs are negligible. Figures 1 and 2 show the run time of the three schemes on 4096 processors. As the figures illustrate, the performance of the different schemes is comparable to the one point per processor case. That is, the FMV iteration usually outperforms the V, and the FMVP scheme slightly outperforms the FMVS iteration. For the most part, the execution times do not vary significantly for the different cycling schemes when there is only one point per processor. However, when each processor contains many points, the FMV cycles significantly outperform the V cycle.

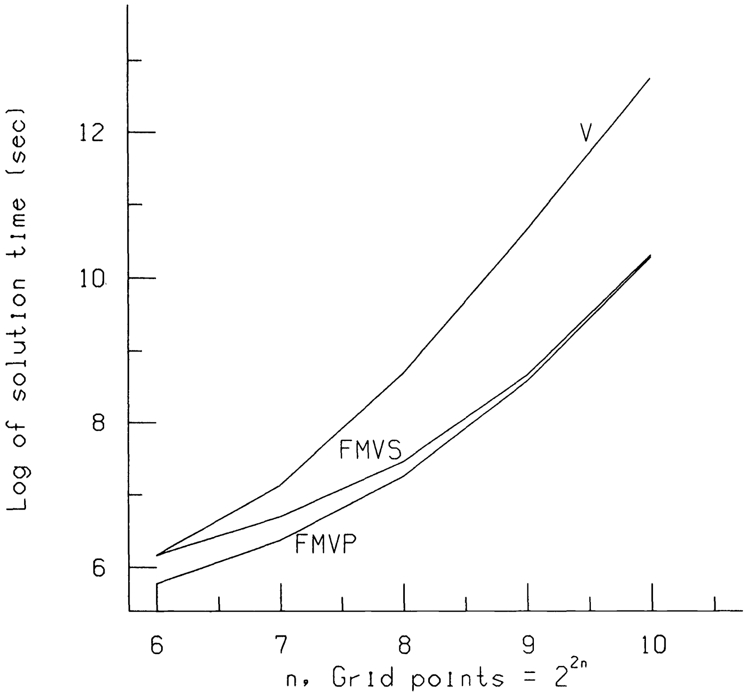


FIG. 1. *Logarithm of execution time for a two-dimensional problem using* V, FMVS, *and* FMVP *cycles on* 4096 *processors assuming zero communication costs on a* 4096 *processor machine.*

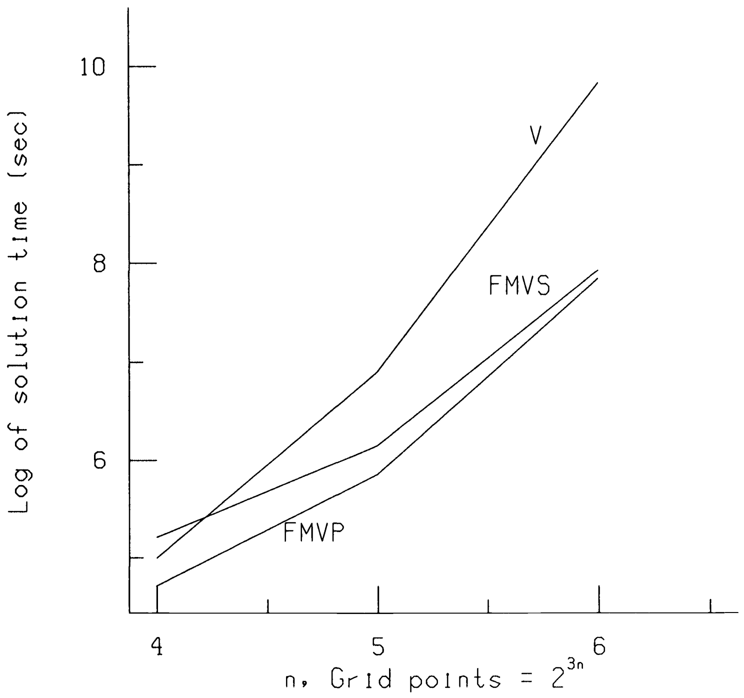


FIG. 2. *Logarithm of execution time for a three-dimensional problem using* V, FMVS, *and* FMVP *cycles assuming zero communication costs on a* 4096 *processor machine.*

Finally, we account for communication costs by considering the following model for the run time on each level:

$$\tau_{i,p}(S) = C_1 + C_2 \left\lceil \frac{N}{P} \right\rceil^{\frac{d-1}{d}} + C_3 \left\lceil \frac{N}{2P} \right\rceil, \tag{12}$$

where the C_i's are determined by doing a least-squares fit to nCUBE 2 timing data for the Gauss–Seidel routine. Specifically, we obtain the following for the nCUBE 2:

$$C_1 = 2.14 \times 10^{-3}, \quad C_2 = 2.02 \times 10^{-5}, \quad C_3 = 4.43 \times 10^{-5}.$$

In Table 3 we can see the close correspondence between the predicted run times using this smoother model and the actual run times obtained on the nCUBE 2 for the smoother.[6]

Using (9), (10), and (12) with the above constants, we obtain the run times and efficiencies for the two- and three-dimensional problems shown in Figs. 3–6. From these plots we can obtain a great deal of information about the performance of the FMV cycle. The first and most important observation is that the FMV cycles significantly outperform the V cycle when each processor contains many grid points, and that the time differential between V and FMV execution times grows quite rapidly as the number of grid points increases. For example, the V cycle calculation on a 1024 × 1024 grid requires almost twice

[6]*Note.* Modelling only the smoother is sufficient because the actual time on each grid level (including projection, interpolation, residual calculation, etc.) is proportional to the smoother time. For our experiments in §4, the actual time per level is about four times larger than the smoother times.

TABLE 3
Predicted and actual nCUBE *run times* (*in seconds*) *of* 1 *red/black Gauss–Seidel* (GS) *iteration.*

GS\Procs	64 × 64 grid 16	256	1024	1024 × 1024 grid 256	1024
model	8.13×10^{-3}	2.57×10^{-3}	2.27×10^{-3}	9.41×10^{-2}	2.56×10^{-2}
nCUBE	8.16×10^{-3}	2.58×10^{-3}	2.28×10^{-3}	9.40×10^{-2}	2.55×10^{-2}

as much time as the corresponding FMVP calculation. Conversely, it is possible that the FMVS scheme may require more time than the V cycle when each processor contains a modest number of points. In these cases, however, the time differential between the V and FMVS schemes is not large and, thus, need not be considered too seriously. Additionally, the FMVP method is the best performer regardless of the number of points per processor. While its performance is better than the FMVS iteration, the time savings between the two FMV methods is not enormous. In our figures, for example, the most significant differences between FMV schemes correspond to the FMVP iteration requiring two-thirds of the time of the FMVS method. Thus, we recommend using the FMVP iteration because it requires the least run time in all cases. However, if it is difficult or inconvenient to compute the optimal grid switching criterion, it is still better to use the standard FMV cycle over the V cycle because it requires substantially less time when there are many points per processor.

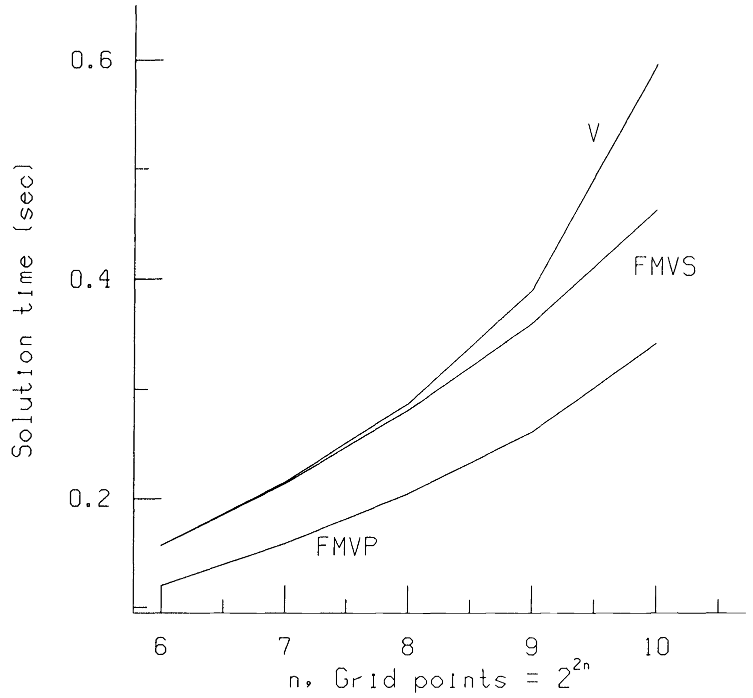


FIG. 3. *Predicted execution time for a two-dimensional problem using* V, FMVS, *and* FMVP *cycles on a* 4096 *processor* nCUBE 2.

Unfortunately, the FMV efficiencies can be extremely low even for large problems. For example, by inspecting Fig. 6 it can be verified that using 4K processors on a 256 ×

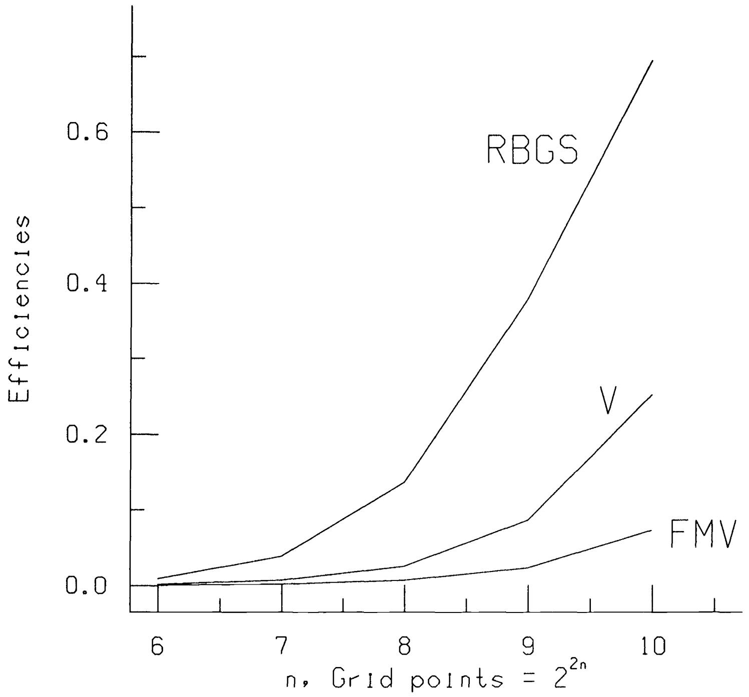


FIG. 4. *Predicted efficiencies for a red/black Gauss–Seidel smoother, a* V *cycle, and an* FMVP *cycle on a two-dimensional problem using a* 4096 *processor* nCUBE 2.

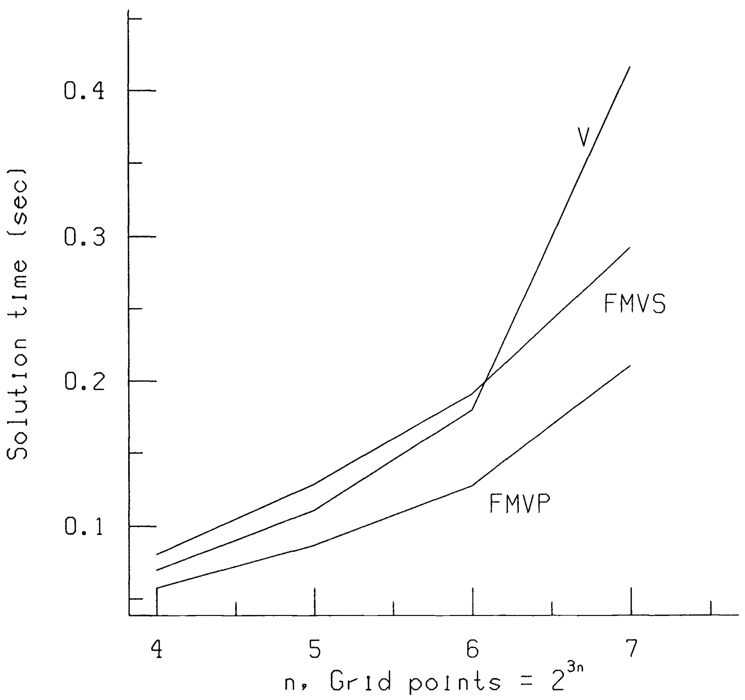


FIG. 5. *Predicted execution time for a three-dimensional problem using the* V, FMVS, *and* FMVP *cycles on a* 4096 *processor* nCUBE 2.

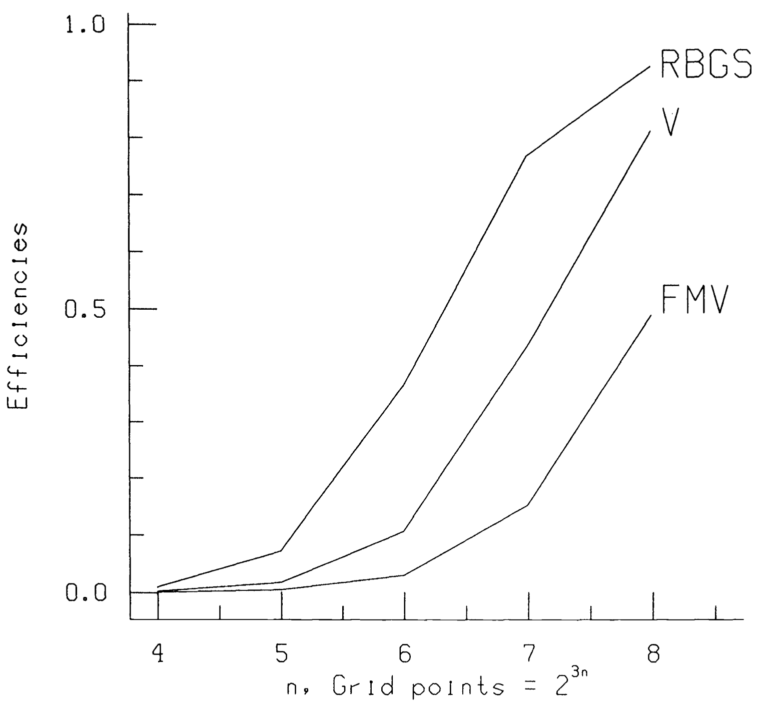


FIG. 6. *Predicted efficiencies for a red/black Gauss–Seidel smoother, a* V *cycle, and an* FMVP *cycle on a three-dimensional problem using a* 4096 *processor* nCUBE 2.

256 × 256 grid yields a smoother that is 92% efficient, a V cycle algorithm that is 82% efficient, and an FMV cycle algorithm that is only 50% efficient. While 50% efficiency is not terrible, a higher efficiency would normally be expected for an explicit algorithm on a grid containing over 16 million points. Note that the drop in efficiency between the smoother and the V cycle is related to the efficiency drop associated with using an FMV cycle as opposed to a V cycle. In fact, the efficiency curves in Figs. 4 and 6 are approximately the same distance apart. For example, in Fig. 6 the smoother runs at 40% efficiency when there are approximately $8^{6.1}$ grid points. Due to the coarse grid computations, the V cycle requires 5.3 times as many points as the smoother to achieve the same parallel efficiency, and the FMV cycle requires 5.3 times as many points as for the V cycle to achieve that same efficiency. The fact that approximately the same increase in points appears is not entirely a coincidence, but is related to the fact that (9) and (10) are identical when the η_i's are the same (which is nearly the case). Though it is somewhat cumbersome to exactly characterize the mathematical relationship between the efficiency curves, we simply state that over a very wide range of communication characteristics (i.e., different C_i's) the same basic relationship holds. That is, the efficiency curves of the smoother, V cycle, and FMV cycle are approximately equidistant. Overall, the FMV efficiencies gradually approach one as the grid becomes larger (similar to other explicit methods). However, it takes many more grid points to achieve these high efficiencies than simple explicit schemes. In our examples, a grid with $42N$ ($28N$) unknowns is necessary to achieve the same efficiency over a wide range of efficiencies as

the smoother applied to N unknowns on a two-dimensional (three-dimensional) grid. That is,

$$\varepsilon_s(N/42) \approx \varepsilon_v(N/7) \approx \varepsilon_{fmvp}(N) \quad \text{in two dimensions}$$

and

$$\varepsilon_s(N/28) \approx \varepsilon_v(N/5.3) \approx \varepsilon_{fmvp}(N) \quad \text{in three dimensions}$$

for the nCUBE hypercube. It is important to remember, however, that despite the poor efficiencies the overall FMV run times are among the fastest for these problems.

We conclude this section by remarking that the characteristics displayed by the various cycles above are seen for a wide range of values of C_1, C_2, and C_3. We also remark that the V cycle and FMVS cycle can be considered special cases of the FMVP cycle, which is optimized for overall solution time. Hence, in the case where communication time dominates computation time (C_1 and C_2 large), the FMVP cycle is the fastest of the three, even though the V cycle may be heavily favored over the FMVS cycle.

7. Implications for parallel processors. While we have focused on two particular multigrid variants, we feel that this study has much more general implications regarding the global mixing of data and the corresponding low efficiencies. In this section, we make a few remarks along these lines.

Within the iterative method community, it has become accepted that the fastest algorithms for elliptic problems require some global mixing of information. That is, the update at a given point should use some information from every other grid point within the domain. Since this global mixing of information requires additional communication and possible load imbalance, the question remains: how much global mixing is desirable on a parallel machine? While it is sometimes stated that a fast iterative method (one which converges independent of mesh size) with a minimal global operations is preferable (e.g., domain decomposition algorithms [7] typically require only one coarse grid problem to be solved), we wish to argue that this reasoning is far too simplistic. That is, there are many situations where the heavy use of global information may be beneficial despite low efficiencies. Clearly, in the multigrid context, the additional use of coarse grids within the FMV cycle is warranted even on massively parallel machines.

Finally, it is really not clear what kind of efficiencies should be expected when solving elliptic PDE systems with "optimal" algorithms. In fact, it can be shown (see [14]) that a lower bound for solving elliptic PDE systems is given by $O(N)$ on serial machines and $O(\log N)$ on parallel machines where N is the number of grid points. Thus, it must be expected that any numerical algorithm that achieves this optimal performance will have an efficiency that drops off sharply when a large number of processors are used. It can be argued that when the FMV cycle is applied to well-behaved systems, the performance is sufficiently close to the theoretical lower bound that the predicted drop in efficiency can be observed. Thus, from this point of view, the low FMV efficiencies should not be that surprising. That is, as one gets closer to an optimal algorithm for solving elliptic problems, lower parallel efficiencies should occur. It is important, however, to realize that in most practical problems the computations are so difficult (or perhaps the algorithms are not sophisticated enough) that the lower bound is not attained and, hence, the sharp drop in efficiency is not observed.

8. Conclusions. We have analyzed the parallel FMV cycle. In particular, a new grid switching criterion was developed to account for parallel processor efficiencies. Using this new criterion, FMV algorithms were implemented and analyzed in a massively parallel setting. Based on this analysis, it was determined that the modified FMV cycle always requires less run time than the corresponding V cycle. This occurs even though the FMV cycle makes far poorer use of parallel processors than the V cycle. Furthermore, the performance of the FMV cycle with the new grid switching criterion is better than that with the standard criterion. However, the overall performance differential is not so great that the user needs to be greatly concerned with implementing the "optimal" switching criterion. In fact, even with the standard switching criterion, the FMV cycle almost always outperforms the V cycle and is within a factor of two of the "optimal" FMV performance. Unfortunately, the overall efficiencies obtained on the nCUBE are somewhat low even on very large grids. While the efficiencies improve as the grid becomes larger, it requires many more grid points to achieve high efficiencies for FMV cycles. Nonetheless, the overall solution times are quite fast and are among the fastest possible with current methods and machines.

Appendix. "Optimal" error ratios between grids. We seek an expression for the ratio of algebraic errors between two consecutive grid levels in an optimal FMV cycle. Consider an FMV cycle on two levels, $l-1$ and l. The algebraic error corresponding to the approximate solution on $\mathcal{G}_{l-1}$ is given by e_{l-1}. On $\mathcal{G}_l$, an approximate solution is obtained from the solution on $\mathcal{G}_{l-1}$ by high-order interpolation. Thus, the algebraic error on level l is given by

$$e_l = e_{l-1} + \delta,$$

where

$$\delta = E_l - E_{l-1}$$

(the difference of the truncation errors). Assume that after k V cycle iterations on $\mathcal{G}_{l-1}$, the norms of the algebraic errors are bounded by $\|\beta^k e_{l-1}\|$ and $\|\beta^k e_{l-1}\| + \|\delta\|$. After k iterations starting from the coarse level, we proceed with j iterations on the fine level resulting in an algebraic error on the fine grid of

$$\beta^j(\beta^k\|e_{l-1}\| + \|\delta\|) \geq \|e_l^{\text{final}}\|. \tag{13}$$

Since we wish to bound the final error, we assume equality in (13) and proceed to determine the k that minimizes the total work necessary to attain a fixed algebraic error bound. Let the cost of a coarse grid V cycle be $\tau_{l-1,p}(V)$ and the cost of a fine grid V cycle be $\xi_{l,p}\tau_{l-1,p}(V)$. Then the total cost is

$$W = (k + j\xi_{l,p})\tau_{l-1,p}(V),$$

where

$$j = \frac{\log\|e_l^{\text{final}}\| - \log(\beta^k\|e_{l-1}\| + \|\delta\|)}{\log(\beta)}.$$

The best choice of k can be found by differentiating W with respect to k and setting the result equal to zero. Namely,

$$0 = 1 - \xi_{l,p}\frac{\beta^k\log(\beta)\|e_{l-1}\|}{\log(\beta)(\beta^k\|e_{l-1}\| + \|\delta\|)}.$$

Thus, we find that the optimal k occurs when

$$\frac{\beta^k \|e_{l-1}\| + \|\delta\|}{\beta^k \|e_{l-1}\|} = \xi_{l,p}.$$

That is, the optimum occurs when the ratio of the algebraic error bounds on the two grids is $\xi_{l,p}$.

REFERENCES

[1] A. BRANDT, *Multigrid techniques*: 1984 *guide, with applications to fluid dynamics*, Tech. Rep. Nr. 85, GMD-AIW, GMD-Studien, St. Augustin, West Germany, 1984.

[2] B. BRIGGS, L. HART, S. MCCORMICK, AND D. QUINLAN, *Multigrid methods on a hypercube*, in Proc. Third Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., Marcel Dekker, New York, 1987, pp. 63–84.

[3] W. BRIGGS, *Multigrid Tutorial*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[4] T. CHAN AND Y. SAAD, *Multigrid algorithms on the hypercube multiprocessor*, IEEE Trans. Comput., C-35 (1986), pp. 969–977.

[5] T. CHAN AND R. TUMINARO, *Design and implementation of parallel multigrid algorithms*, in Proc. the Third Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., Marcel Dekker, New York, 1987, pp. 101–115.

[6] ———, *A survey of parallel multigrid algorithms*, in Proc. ASME Symp. Parallel Computations and their Impact on Mechanics, A. Noor, ed., Vol. AMD, Vol. 86, American Society of Mechanical Engineers, 1987, pp. 155–170.

[7] R. GLOWINSKI, G. GOLUB, G. MEURANT, AND J. PERIAUX, EDS., *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[8] O. MCBRYAN AND E. V. DE VELDE, *Multigrid method on parallel processors*, in Multigrid Methods II, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Math. No. 1228, Springer-Verlag, Berlin, 1986, pp. 232–260.

[9] C. THOLE, *Experiments with multigrid on the Caltech hypercube*, Tech. Rep. Nr. 103, GMD-Studien, St. Augustin, West Germany, 1985.

[10] ———, *A short note on parallel multigrid algorithms for* 3d-*problems*, Tech. Rep. Nr. 102, GMD-Studien, St. Augustin, West Germany, 1987.

[11] R. TUMINARO, *Multigrid Algorithms on Parallel Processing Systems*, Ph.D. thesis, Stanford University, Stanford, CA, 1989.

[12] S. VANDEWALLE AND R. PIESSENS, *A comparison of parallel multigrid strategies*, Tech. Rep. TW 120, Katholieke Universiteit Leuven, Computer Science Dept., Leuven, Belgium, February 1989.

[13] D. WOMBLE AND B. YOUNG, *A model and implementation of multigrid for massively parallel computers*, Internat. J. High Speed Comput., 2 (1990), pp. 239–255.

[14] P. WORLEY, *Problem Dissection and the Implications for Parallel Computation*, Ph.D. thesis, Stanford University, Stanford, CA, 1988.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1174-1193, September 1993

FAST INVERSE QR FACTORIZATION FOR TOEPLITZ MATRICES*

JAMES G. NAGY†

Abstract. Fast orthogonalization schemes for $m \times n$ Toeplitz matrices T, introduced by Bojanczyk, Brent, and de Hoog (BBH) and Chun, Kailath, and Lev-Ari (CKL), are extended to compute directly an inverse QR factorization of T using only $O(mn)$ operations. An inverse factorization allows for an efficient parallel implementation, and the algorithm is computationally less expensive for computing a solution to the Toeplitz least squares problem than previously studied inverse QR methods. In addition, it is shown that regularization can be incorporated into the algorithm with virtually no extra work. Thus it is possible to compute regularized solutions to ill-conditioned Toeplitz least squares problems using only $O(mn)$ operations. An application to ill-conditioned problems occurring in signal restoration is provided, illustrating the effectiveness of this method.

Key words. least squares problem, QR factorization, Toeplitz matrix, regularization

AMS subject classifications. 65F05, 65F25

1. Introduction.

An $m \times n$ matrix T is a Toeplitz matrix if its entries are constant down each diagonal. This can be expressed by denoting the (i, j) entry of T as t_{i-j}. In this paper we will consider solving least squares (LS) problems of the form

$$\text{minimize } \|b - Tx\|_2, \tag{1.1}$$

where T is an $m \times n$ Toeplitz matrix of full column rank, b is an $m \times 1$ vector and $\|\cdot\|_2$ denotes the usual Euclidean norm. Toeplitz least squares problems occur in many engineering applications such as signal restoration [12], data compression [12], seismic exploration [14], and speech echo cancellation [2].

Least squares problems of the form (1.1) can be solved effectively using the QR factorization of T, which can be computed using $O(mn^2)$ operations. By exploiting the Toeplitz structure of T, we can obtain a *fast* QR factorization which requires only $O(mn)$ operations. This is the case for the algorithms of Sweet [18], Bojanczyk, Brent, and de Hoog [5], Chun, Kailath, and Lev-Ari [8], and Cybenko [9]. It also has been proposed recently to use the preconditioned conjugate gradient (PCG) algorithm to solve Toeplitz least squares problems [7], [15], [16]. For certain Toeplitz matrices it is shown that the PCG method is effective in solving these LS problems.

The first $O(mn)$ method for computing the QR factorization of a Toeplitz matrix was developed by Sweet [18], which he has more recently extended to block Toeplitz matrices [19]. Bojanczyk, Brent, and de Hoog [5] modified the ideas of Sweet to develop a slightly faster QR factorization. (We will denote the fast QR factorization of Bojanczyk, Brent, and de Hoog by BBH.) Chun, Kailath, and Lev-Ari introduced a fast, and more general, QR factorization algorithm based on the displacement structure of T and T^TT. (We denote this method by CKL.) Although they use a different approach, their method is essentially equivalent to that of BBH. Moreover, in the CKL algorithm, certain parameters generated in obtaining R can also be used to compute R^{-1} in $O(n^2)$ operations. Finally, a different orthogonalization scheme for Toeplitz matrices was proposed by Cybenko [9]. This algorithm uses inner products to construct Q and R^{-1}, but requires the computation of Q to obtain R^{-1}.

*Received by the editors December 9, 1991; accepted for publication (in revised form) November 2, 1992.

†Department of Mathematics, Southern Methodist University, Dallas, Texas 75275. This work was completed while the author was visiting the Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, Minnesota.

In §2 of this paper we describe the method of BBH for computing R and show in §3 how it can be modified, using an inverse factorization scheme by Pan and Plemmons [17], to compute R^{-1} directly. Our algorithm does not require the computation of R to obtain R^{-1}, and hence is less expensive than the CKL method. In §4 we then combine this with the CKL approach for computing Q, to obtain a fast inverse QR factorization algorithm. Moreover, we show that Q^Tb can easily be computed without explicitly forming Q. This provides considerable savings in the computational work in obtaining the LS solution, which cannot be attained in Cybenko's algorithm. Finally, in §5 we consider ill-conditioned problems that occur often in signal processing applications. We use the method of regularization to alleviate the problems caused by the ill conditioning, and show that the fast orthogonalization schemes discussed here extend nicely for these problems. Some numerical examples are provided in §6.

2. Computation of R: Algorithm BBH-R.

Let T be an $m \times n$ Toeplitz matrix of full column rank, and let R be the Cholesky factor of T^TT. In this section we give a brief description of how the BBH algorithm constructs R one row at a time.

Since T has the Toeplitz structure, we can partition T as

$$T = \begin{bmatrix} t_0 & u^T \\ v & T_0 \end{bmatrix} = \begin{bmatrix} T_0 & \tilde{u} \\ \tilde{v}^T & t_{m-n} \end{bmatrix}, \tag{2.1}$$

where T_0 is a submatrix of T, u and $\tilde{v}$ are $n-1$ dimensional vectors, v and $\tilde{u}$ are $m-1$ dimensional vectors, and t_0 and t_{m-n} are scalars. In addition, consider partitioning R as

$$R = \begin{bmatrix} r_{11} & z^T \\ 0 & R_b \end{bmatrix} = \begin{bmatrix} R_t & \tilde{z} \\ 0^T & r_{nn} \end{bmatrix}, \tag{2.2}$$

where z and $\tilde{z}$ are $n-1$ dimensional vectors and r_{11} and r_{nn} are scalars.

Setting $R^TR = T^TT$ and using the partitionings (2.1) and (2.2), we have

$$\begin{bmatrix} r_{11}^2 & r_{11}z^T \\ r_{11}z & zz^T + R_b^TR_b \end{bmatrix} = \begin{bmatrix} t_0^2 + v^Tv & t_0u^T + v^TT_0 \\ t_0u + T_0^Tv & uu^T + T_0^TT_0 \end{bmatrix} \tag{2.3}$$

and

$$\begin{bmatrix} R_t^TR_t & R_t^T\tilde{z} \\ \tilde{z}^TR_t & \tilde{z}^T\tilde{z} + r_{nn}^2 \end{bmatrix} = \begin{bmatrix} T_0^TT_0 + \tilde{v}\tilde{v}^T & T_0^T\tilde{u} + t_{m-n}\tilde{v} \\ \tilde{u}^TT_0 + t_{m-n}\tilde{v}^T & \tilde{u}^T\tilde{u} + t_{m-n}^2 \end{bmatrix}. \tag{2.4}$$

From (2.3) and (2.4) we see that

$$zz^T + R_b^TR_b = uu^T + T_0^TT_0 \quad \text{and} \quad R_t^TR_t = T_0^TT_0 + \tilde{v}\tilde{v}^T.$$

Combining the above two relations, we obtain

$$R_b^TR_b = R_t^TR_t + uu^T - \tilde{v}\tilde{v}^T - zz^T. \tag{2.5}$$

Observe also from (2.3) that $r_{11}^2 = t_0^2 + v^Tv$ and $z^T = (t_0u^T + v^TT_0)/r_{11}$, and so the first row of R can easily be computed.

Equation (2.5) is used to compute R one row at a time as follows. First rewrite (2.5) as

$$R_1^T R_1 = R_t^T R_t + uu^T, \tag{2.6}$$

$$R_2^T R_2 = R_1^T R_1 - \tilde{v}\tilde{v}^T, \tag{2.7}$$

and

$$R_b^T R_b = R_2^T R_2 - zz^T, \tag{2.8}$$

where R_1 and R_2 are upper triangular matrices. Notice that (2.6) is an updating problem and (2.7) and (2.8) are downdating problems.

It is well known (see, for example, [11]) that, for the updating problem, we can find a product of Givens rotations $Q = G(n-1, n, \theta_{n-1}) \cdots G(1, n, \theta_1)$ so that

$$Q \begin{bmatrix} R_t \\ u^T \end{bmatrix} = \begin{bmatrix} R_1 \\ 0^T \end{bmatrix}, \tag{2.9}$$

where $G(i, j, \theta_i)$ is an elementary Givens rotation; that is, a matrix which is the identity everywhere except $g_{ii} = g_{jj} = c$ and $g_{ji} = -g_{ij} = s$, where $c = c(\theta) = \cos(\theta)$ and $s = s(\theta) = \sin(\theta)$ for some θ. We will use the notation

$$[c, s] = [c(\theta), s(\theta)] = \text{givens}(\alpha, \beta)$$

to be the 2×2 Givens rotation for which

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \bar{\alpha} \\ 0 \end{bmatrix}.$$

For the downdating problems, we attempt to find products of Hyperbolic rotations $Y^{(1)} = H(n-1, n, \phi_{n-1}) \cdots H(1, n, \phi_1)$ and $Y^{(2)} = H(n-1, n, \rho_{n-1}) \cdots H(1, n, \rho_1)$ such that

$$Y^{(1)} \begin{bmatrix} R_1 \\ \tilde{v}^T \end{bmatrix} = \begin{bmatrix} R_2 \\ 0^T \end{bmatrix} \tag{2.10}$$

and

$$Y^{(2)} \begin{bmatrix} R_2 \\ z^T \end{bmatrix} = \begin{bmatrix} R_b \\ 0^T \end{bmatrix}. \tag{2.11}$$

It can be shown that if the downdated matrices are nonsingular then such products exist [2]. The matrix $H(i, j, \phi)$, called an elementary Hyperbolic rotation, is a matrix which is the identity everywhere except $h_{ii} = h_{jj} = c$ and $h_{ij} = h_{ji} = -s$, where $c = c(\phi) = \cosh(\phi)$ and $s = s(\phi) = \sinh(\phi)$ for some ϕ. As with Givens rotations, we will use the notation

$$[c, s] = [c(\phi), s(\phi)] = \text{hyp}(\alpha, \beta)$$

to be the 2×2 Hyperbolic rotation for which

$$\begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \bar{\alpha} \\ 0 \end{bmatrix}.$$

Equations (2.9), (2.10), and (2.11) provide a method for computing R one row at a time. This can be done by observing that for each i, $G(i, n, \theta_i)$, $H(i, n, \phi_i)$ and $H(i, n, \rho_i)$ affect only rows i and n. Since the first row of R_t is (modulo a shift) the same as the first row of R, we know the first row of R_t. Thus $G(1, n, \theta_1)$ can be applied to the first row of R_t and u^T to obtain the first row of R_1. Since, now, the first row of R_1 is known, we can apply $H(1, n, \phi_1)$ to the first row of R_1 and $\tilde{v}^T$ to obtain the first row of R_2. Thus since the first row of R_2 is now known, we can apply $H(1, n, \rho_1)$ to the first row of R_2 and z^T to obtain the first row of R_b.

Observe from (2.2) that the first row of R_b is the second row of R, which is (modulo a shift) the second row of R_t. Thus we repeat the above process with $G(2, n, \theta_2)$, $H(2, n, \phi_2)$, and $H(2, n, \rho_2)$, etc.

The following algorithm summarizes the above results, where r_t, r_1, r_2, and r_b are vectors used in intermediate steps of the algorithm.

Algorithm BBH-R. *Let T be an $m \times n$ Toeplitz matrix, and let u, v, and $\tilde{v}$ be defined as in* (2.1). *Then this algorithm computes the Cholesky factor, R, of $T^T T$.*

```
$R(1,1) = r_{11} = \sqrt{t_0^2 + t_1^2 + \cdots + t_{m-1}^2}$
$z = (t_0 u + T_0^T v)/r_{11}$
$R(1, 2:n) = z^T$
for $k = 1, 2, \ldots, n-1$
    $r_t(k:n-1) = R(k, k:n-1)$
    $[c, s] =$ givens$(r_t(k), u(k))$
    $\begin{bmatrix} r_1^T \\ u^T \end{bmatrix} := \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} r_t^T \\ u^T \end{bmatrix}$
    $[c, s] =$ hyp$(r_1(k), \tilde{v}(k))$
    $\begin{bmatrix} r_2^T \\ \tilde{v}^T \end{bmatrix} := \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} r_1^T \\ \tilde{v}^T \end{bmatrix}$
    $[c, s] =$ hyp$(r_2(k), z(k))$
    $\begin{bmatrix} r_b^T \\ z^T \end{bmatrix} := \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} r_2^T \\ z^T \end{bmatrix}$
    $R(k+1, k+1:n) = r_b(k:n-1)$.
```

Algorithm BBH-R requires $mn + 6n^2 + O(n)$ multiplications to compute R. We remark here that the CKL method for computing R also requires $mn + 6n^2 + O(n)$ multiplications but can also compute R^{-1} using only an additional $O(n^2)$ multiplications [8]. In the next section we will use the approach of BBH to develop a scheme to compute R^{-1} which does not require computing R, and hence is computationally less expensive than the CKL method.

3. Computation of R^{-1}: Algorithm BBH-L. In this section we show how Algorithm BBH-R can be modified to compute R^{-T}, rather than R. The modification is based on an inverse factorization scheme by Pan and Plemmons [17]. The results we need from

[17] can be stated as follows. Let R_0 be an $(n-1)\times(n-1)$ upper triangular matrix and let g be an $n-1$ dimensional vector. Suppose Q is a product of Givens rotations such that

$$Q\begin{bmatrix} R_0 & 0 \\ g^T & 1 \end{bmatrix} = \begin{bmatrix} \tilde{R}_0 & q \\ 0^T & \kappa \end{bmatrix}, \tag{3.1}$$

where $\tilde{R}_0$ is an upper triangular matrix. Assuming R_0 is nonsingular, then it follows that $\kappa \neq 0$. Inverting and transposing both sides of (3.1), we obtain

$$Q\begin{bmatrix} R_0^{-T} & -a \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} \tilde{R}_0^{-T} & 0 \\ h^T & \delta \end{bmatrix}, \tag{3.2}$$

where $a = R_0^{-T}g$, $h = -\tilde{R}_0^{-1}q/\kappa$, and $\delta = 1/\kappa$. By equating the last columns in (3.2), it follows that Q can be determined from a by finding the product of Givens rotations, $Q = G(n-1, n, \theta_{n-1})\cdots(1, n, \theta_1)$, such that

$$Q\begin{bmatrix} -a \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \delta \end{bmatrix}.$$

Moreover, since Q is an orthogonal matrix, it is easy to see that $\delta = \sqrt{1 + \|a\|_2^2}$.

Similarly, if Y is a product of Hyperbolic rotations such that

$$Y\begin{bmatrix} R_0 & 0 \\ g^T & 1 \end{bmatrix} = \begin{bmatrix} \tilde{R}_0 & q \\ 0^T & \kappa \end{bmatrix},$$

then

$$Y\begin{bmatrix} R_0^{-T} & a \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} \tilde{R}_0^{-T} & 0 \\ -h^T & \gamma \end{bmatrix},$$

where $a = R_0^{-T}g$ and $\gamma = \sqrt{1 - \|a\|_2^2}$. In this case, Y can be determined by finding the product of Hyperbolic rotations $Y = H(n-1, n, \phi_{n-1})\cdots H(1, n, \phi_1)$ such that

$$Y\begin{bmatrix} a \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \gamma \end{bmatrix}.$$

Recall that in the last section we obtained the relations

$$Q\begin{bmatrix} R_t \\ u^T \end{bmatrix} = \begin{bmatrix} R_1 \\ 0^T \end{bmatrix}, \quad Y^{(1)}\begin{bmatrix} R_1 \\ \tilde{v}^T \end{bmatrix} = \begin{bmatrix} R_2 \\ 0^T \end{bmatrix}, \quad Y^{(2)}\begin{bmatrix} R_2 \\ z^T \end{bmatrix} = \begin{bmatrix} R_b \\ 0^T \end{bmatrix}.$$

It follows from the above remarks that

$$Q\begin{bmatrix} R_t^{-T} \\ 0^T \end{bmatrix} = \begin{bmatrix} R_1^{-T} \\ h_1^T \end{bmatrix}, \quad Y^{(1)}\begin{bmatrix} R_1^{-T} \\ 0^T \end{bmatrix} = \begin{bmatrix} R_2^{-T} \\ h_2^T \end{bmatrix},$$

$$Y^{(2)}\begin{bmatrix} R_2^{-T} \\ 0^T \end{bmatrix} = \begin{bmatrix} R_b^{-T} \\ h_3^T \end{bmatrix},$$

for some vectors h_1, h_2, and h_3.

Thus to compute R^{-T} one row at a time, we will need to (i) relate the rows of R^{-T} to the rows of R_t^{-T} and R_b^{-T} and (ii) compute the first row of R^{-T}. To accomplish this, we rewrite (2.2) as

$$R^{-T} = \begin{bmatrix} 1/r_{11} & 0^T \\ -R_b^{-T} z/r_{11} & R_b^{-T} \end{bmatrix} = \begin{bmatrix} R_t^{-T} & 0 \\ -\tilde{z}^T R_t^{-T}/r_{nn} & 1/r_{nn} \end{bmatrix}. \tag{3.3}$$

From (3.3) it is easy to see that the first row of $R^{-T} = [\ 1/r_{11} \ \ 0 \ \cdots \ 0\ ]$ and that the rows of R_t^{-T} are (essentially) the same as those of R^{-T}. Furthermore, if the kth row of R_t^{-T}, say l_t^T, is known, then $G(k, n-1, \theta_k)$ can be found using the following steps.

- Set (the scalar) $\alpha = l_t^T u$,
- $[c, s]$ =givens$(\delta_{k-1}, -\alpha)$,
- $\delta_k = -s a + c\delta_{k-1}$,

where $\delta_1 = 1$. $H(k, n-1, \phi_k)$ and $H(k, n-1, \rho_k)$ are found in a similar manner.

Now, applying $G(k, n-1, \theta_k)$, $H(k, n-1, \phi_k)$, and $H(k, n-1, \rho_k)$ to the kth rows of R_t^{-T}, R_1^{-T}, and R_2^{-T}, respectively, we obtain the kth rows of R_1^{-T}, R_2^{-T}, and R_b^{-T}, respectively. Moreover, if we denote the kth row of R_b^{-T} by l_b^T, then, from the first partitioning in (3.3), we see that the $(k+1)$st row of $L = R^{-T}$ is

$$L(k+1, 1:k+1) = \begin{bmatrix} -l_b^T z/r_{11} & l_b^T \end{bmatrix}.$$

Putting the above results together, we obtain the following algorithm for computing R^{-T}.

Algorithm BBH-L. *Let T be an $m \times n$ Toeplitz matrix, and let u, v, and $\tilde{v}$ be defined as in* (2.1). *Then this algorithm computes $L = R^{-T}$, where R is the Cholesky factor of $T^T T$.*

```
$r_{11} = \sqrt{t_0^2 + t_1^2 + \cdots + t_{m-1}^2}$
$z = (t_0 u + T_0^T v)/r_{11}$, $L(1,1) = 1/r_{11}$
$\delta = \gamma_1 = \gamma_2 = 1$, $h_1^T = h_2^T = h_3^T = 0^T$
for $k = 1, 2, \ldots, n-1$
    $l_t(1:k) = L(k, 1:k)$
    $\alpha = l_t^T u(1:k)$
    $[c, s]$ =givens$(\delta, -\alpha)$
    $\delta = s\alpha + c\delta$
    $\begin{bmatrix} l_1^T \\ h_1^T \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} l_t^T \\ h_1^T \end{bmatrix}$
    $\beta = l_1^T \tilde{v}(1:k)$
    $[c, s]$ =hyp$(\gamma_1, \beta)$
    $\gamma_1 = -s\beta + c\gamma_1$
    $\begin{bmatrix} l_2^T \\ h_2^T \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} l_1^T \\ h_2^T \end{bmatrix}$
    $\beta = l_2^T z(1:k)$
    $[c, s]$ =hyp$(\gamma_2, \beta)$
    $\gamma_2 = -s\beta + c\gamma_2$
    $\begin{bmatrix} l_b^T \\ h_3^T \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} l_2^T \\ h_3^T \end{bmatrix}$
    $L(k+1, 1:k+1) = \begin{bmatrix} -l_b^T z/r_{11} & l_b^T \end{bmatrix}$.
```

Algorithm BBH-L requires $mn + 8n^2 + O(n)$ multiplications to compute R^{-1}, whereas the CKL method requires $mn + 12n^2 + O(n)$ multiplications.

4. Computation of Q and Q^Tb. In this section we describe the CKL method for computing the QR factorization of T and show that this method for computing R is equivalent to Algorithm BBH-R. It will then follow that the elementary Givens and Hyperbolic rotations, used in the previous section to compute R^{-T}, can be used to obtain Q and Q^Tb.

Let T and R have the partitionings given in (2.1) and (2.2), respectively, and define $U(w) \in \Re^{n\times k}$ to be the upper triangular Toeplitz matrix with first row $w^T \in \Re^k$.

LEMMA 4.1. *Let T be a Toeplitz matrix and let R be the Cholesky factor of T^TT. Let $w_1^T = [\, r_{11} \quad z^T \,]$, $w_2^T = [\, 0 \quad u^T \,]$, $w_3^T = [\, 0 \quad \tilde{v}^T \,]$, and $w_4^T = [\, 0 \quad z^T \,]$, where z, u, and $\tilde{v}$ are defined from the partitionings* (2.1) *and* (2.2). *Then*

$$T^TT = \sum_{i=1}^{2} U^T(w_i)U(w_i) - \sum_{i=3}^{4} U^T(w_i)U(w_i). \tag{4.1}$$

Proof. For the proof, see [8, Lemma 2]. □

The right-hand side of (4.1) is known as the displacement representation of T^TT [8].

It can be shown that the triangular factors $U(w_i)$ can be used to obtain R as follows. Define the matrix $\Gamma \in \Re^{4n\times n}$ as

$$\Gamma = \begin{bmatrix} U(w_1) \\ U(w_2) \\ U(w_3) \\ U(w_4) \end{bmatrix} \tag{4.2}$$

and let $S \in \Re^{4n\times 4n}$ be defined as $S = \mathrm{diag}(I_n, I_n, \tilde{I}_n, \tilde{I}_n)$, where $\tilde{I}_n = \left[\begin{smallmatrix} -I_{n-1} & 0 \\ 0^T & 1 \end{smallmatrix}\right]$. Then a matrix $B \in \Re^{4n\times 4n}$ is said to be S-orthogonal if $B^TSB = S$.

THEOREM 4.2. *Suppose Θ is an S-orthogonal matrix such that*

$$\tilde{\Gamma} = \Theta\Gamma = \begin{bmatrix} R \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where R is upper triangular. Then $R^TR = T^TT$.

Proof. For the proof, see [8, Thm. 2]. □

We will now show how a particular Θ can be constructed from the elementary Givens and Hyperbolic rotations used in §3 to compute R^{-T}. Recall that Q, $Y^{(1)}$, and $Y^{(2)}$ are defined as

- $Q = G(n-1, n, \theta_{n-1}) \cdots G(2, n, \theta_2)G(1, n, \theta_1)$,
- $Y^{(1)} = H(n-1, n, \phi_{n-1}) \cdots H(2, n, \phi_2)H(1, n, \phi_1)$,
- $Y^{(2)} = H(n-1, n, \rho_{n-1}) \cdots H(2, n, \rho_2)H(1, n, \rho_1)$,

where $G(i, j, \theta_i)$, $H(i, j, \phi_i)$, and $H(i, j, \rho_i)$ are elementary Givens, Hyperbolic, and Hyperbolic rotations, respectively.

Now notice that the matrices $U(w_i)$ are determined by the vectors u, $\tilde{v}$, and z. Furthermore, the first row of R_t is the same (modulo a shift) as the second row of $U(w_1)$. This observation is the motivation for the following theorem.

THEOREM 4.3. *Let* Γ *be defined as in* (4.2), *and let*

$$\Theta = \tilde{H}^{(2)}_{n-1}\tilde{H}^{(1)}_{n-1}\tilde{G}_{n-1}\cdots\tilde{H}^{(2)}_{2}\tilde{H}^{(1)}_{2}\tilde{G}_2\tilde{H}^{(2)}_{1}\tilde{H}^{(1)}_{1}\tilde{G}_1, \tag{4.3}$$

where

$$\begin{aligned}
\tilde{G}_k &= G(n, 2n-k, \theta_k)\cdots G(k+2, n+2, \theta_k)G(k+1, n+1, \theta_k),\\
\tilde{H}^{(1)}_k &= H(n, 3n-k, \phi_k)\cdots H(k+2, 2n+2, \phi_k)H(k+1, 2n+1, \phi_k),\\
\tilde{H}^{(2)}_k &= H(n, 4n-k, \rho_k)\cdots H(k+2, 3n+2, \rho_k)H(k+1, 3n+1, \rho_k).
\end{aligned}$$

Then

$$\Theta\Gamma = \begin{bmatrix} R \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where R *is upper triangular.*

Proof. Let R_t, R_1, R_2, and R_b be defined as in §2, and recall that the Givens rotations $G(k, n, \theta_k)$ were constructed so that

$$\begin{bmatrix} k\text{th row of } R_1 \\ u^T \end{bmatrix} := \begin{bmatrix} c(\theta_k) & -s(\theta_k) \\ s(\theta_k) & c(\theta_k) \end{bmatrix} \begin{bmatrix} k\text{th row of } R_t \\ u^T \end{bmatrix}, \tag{4.4}$$

where the emphasis on θ_k is used to indicate which rotation is used. The vector u^T on the right-hand side of (4.4) has the form

$$[\underbrace{0 \cdots 0}_{k-1} * \cdots *]$$

and on the left-hand side u^T has the form

$$[\underbrace{0 \cdots 0\, 0}_{k} * \cdots *],$$

where $*$ indicates an element which may not be 0. Similar relations hold for the Hyperbolic rotations $H(k, n, \phi_k)$ and $H(k, n, \rho_k)$.

Observe that

$$\begin{bmatrix} \text{2nd row of } U(w_1) \\ \text{1st row of } U(w_2) \end{bmatrix} = \begin{bmatrix} 0 & \text{1st row of } R_t \\ 0 & u^T \end{bmatrix}.$$

From the above remarks, we know that $G(1, n, \theta_1)$ is constructed so that

$$\begin{bmatrix} c(\theta_1) & -s(\theta_1) \\ s(\theta_1) & c(\theta_1) \end{bmatrix} \begin{bmatrix} 0 & \text{1st row of } R_t \\ 0 & u^T \end{bmatrix} =: \begin{bmatrix} 0 & \text{1st row of } R_1 \\ 0 & u^T \end{bmatrix},$$

where u^T has been overwritten and now has the form $[\,0\ * \cdots *\,]$.

The Toeplitz structure of $U(w_1)$ and $U(w_2)$ implies that $c(\theta_1)$ and $s(\theta_1)$ can be used to zero out the first super diagonal of $U(w_2)$. This is equivalent to applying the product of Givens rotations

$$\tilde{G}_1 = G(n, 2n-1, \theta_1) \cdots G(3, n+2, \theta_1) G(2, n+1, \theta_1)$$

to Γ.

Assume Γ is overwritten with $\tilde{G}_1\Gamma$. Then observe that

$$\begin{bmatrix} \text{2nd row of } U(w_1) \\ \text{1st row of } U(w_3) \end{bmatrix} = \begin{bmatrix} 0 & \text{1st row of } R_1 \\ 0 & \tilde{v}^T \end{bmatrix}.$$

We know that $H(1, n, \phi_1)$ is constructed so that

$$\begin{bmatrix} c(\phi_1) & -s(\phi_1) \\ -s(\phi_1) & c(\phi_1) \end{bmatrix} \begin{bmatrix} 0 & \text{1st row of } R_1 \\ 0 & \tilde{v}^T \end{bmatrix} =: \begin{bmatrix} 0 & \text{1st row of } R_2 \\ 0 & \tilde{v}^T \end{bmatrix},$$

where $\tilde{v}^T$ has been overwritten and now has the form $[\, 0 \ * \cdots * \,]$.

The Toeplitz structure of $U(w_1)$ and $U(w_3)$ implies that $c(\phi_1)$ and $s(\phi_1)$ can be used to zero out the first super diagonal of $U(w_3)$. This is equivalent to applying the product of Hyperbolic rotations

$$\tilde{H}_1^{(1)} = H(n, 3n-1, \phi_1) \cdots H(3, 2n+2, \phi_1) H(2, 2n+1, \phi_1)$$

to $\Gamma = \tilde{G}_1\Gamma$.

Assume Γ is overwritten with $\tilde{H}_1^{(1)}\tilde{G}_1\Gamma$. Then observe that

$$\begin{bmatrix} \text{2nd row of } U(w_1) \\ \text{1st row of } U(w_4) \end{bmatrix} = \begin{bmatrix} 0 & \text{1st row of } R_2 \\ 0 & z^T \end{bmatrix}.$$

We know that $H(1, n, \rho_1)$ is constructed so that

$$\begin{bmatrix} c(\rho_1) & -s(\rho_1) \\ -s(\rho_1) & c(\rho_1) \end{bmatrix} \begin{bmatrix} 0 & \text{1st row of } R_2 \\ 0 & z^T \end{bmatrix} =: \begin{bmatrix} 0 & \text{1st row of } R_b \\ 0 & z^T \end{bmatrix},$$

where z^T has been overwritten and now has the form $[\, 0 \ * \cdots * \,]$.

The Toeplitz structure of $U(w_1)$ and $U(w_4)$ implies that $c(\rho_1)$ and $s(\rho_1)$ can be used to zero out the first super diagonal of $U(w_4)$. This is equivalent to applying the product of Hyperbolic rotations

$$\tilde{H}_1^{(2)} = H(n, 4n-1, \rho_1) \cdots H(3, 3n+2, \rho_1) H(2, 3n+1, \rho_1)$$

to $\Gamma = \tilde{H}_1^{(1)}\tilde{G}_1\Gamma$.

Now, assuming Γ has been overwritten with $\tilde{H}_1^{(2)}\tilde{H}_1^{(1)}\tilde{G}_1\Gamma$, we have

$$\text{2nd row of } U(w_1) = [\, 0 \ \ \text{1st row of } R_b \,] = \text{2nd row of } R.$$

Thus the first two rows of $U(w_1)$ are the same as those in R. Furthermore, we have zeros in all the super diagonals of $U(w_2)$, $U(w_3)$, and $U(w_4)$.

Because of the Toeplitz structure of $U(w_1)$, we have

$$\begin{bmatrix} \text{3rd row of } U(w_1) \\ \text{2nd row of } U(w_2) \end{bmatrix} = \begin{bmatrix} 0 & \text{2nd row of } R_t \\ 0 & u^T \end{bmatrix}.$$

It is now easy to see that, by repeating the above arguments for $G(2,n,\theta_2)$, $H(2,n,\phi_2)$, and $H(2,n,\rho_2)$ we will zero out the second super diagonals of $U(w_2)$, $U(w_3)$, and $U(w_4)$ and obtain the first three rows of R in $U(w_1)$.

Thus by repeating the above arguments for $G(k,n,\theta_k)$, $H(k,n,\phi_k)$, and $H(k,n,\rho_k)$ the result follows. □

THEOREM 4.4. *Let Θ be defined as in* (4.3). *Then Θ is S-orthogonal.*

Proof. We show first that for each k, $\tilde{G}_k$, $\tilde{H}_k^{(1)}$, and $\tilde{H}_k^{(2)}$ are all S-orthogonal. Now $\tilde{G}_k$ is an orthogonal matrix which only affects rows 2 through $2n-1$. Thus we can write

$$\tilde{G}_k = \begin{bmatrix} \tilde{Q} & & \\ & I_n & \\ & & I_n \end{bmatrix},$$

where $\tilde{Q}^T\tilde{Q} = I_{2n}$. Using this observation it is easy to see that $\tilde{G}_k^T S \tilde{G}_k = S$, and hence $\tilde{G}_k$ is S-orthogonal.

To show that each $\tilde{H}_k^{(1)}$ is S-orthogonal requires a little more work. First, we define S_j to be the identity matrix except that the (j,j) entry is -1. Then notice that $H(i,l,\phi)S_j = S_j H(i,l,\phi)$, provided $j \neq i,l$. Also observe that

$$H^T(k+j,n+j,\phi_k)S_{n+j}H(k+j,n+j,\phi_k) = S_{n+j}.$$

From the commutative property stated above, it follows that each $\tilde{H}_k^{(1)}$ is $\hat{S}$-orthogonal where $\hat{S}$ =diag$(I_n, I_n, \bar{I}_n, I_n)$. But since $\tilde{H}_k^{(1)}$ does not affect rows $3n+1$ through $4n$, it follows that $\tilde{H}_k^{(1)}$ is S-orthogonal.

A similar argument holds for $\tilde{H}_k^{(2)}$, and since the product of S-orthogonal matrices is S-orthogonal, we have that Θ is S-orthogonal. □

Since we already have an algorithm that computes R^{-T}, the preceding results may seem redundant. But, as we will shortly see, a transformation Θ which satisfies Theorem 4.2 can be used to obtain Q and Q^Tb.

LEMMA 4.5. *Let T be an $m \times n$ Toeplitz matrix, and let $a \in \Re^m$ be the first column of T. If $y_1 = a/\|a\|_2 = -y_4$, $y_2 = e_1$, $y_3 = 0$, and w_i are defined as in Lemma* 4.1, *then*

$$T = \sum_{i=1}^{4} U^T(y_i)U(w_i).$$

Proof. For the proof, see [8, Lemma 6]. □

THEOREM 4.6. *If Θ is an S-orthogonal matrix that satisfies Theorem* 4.2, *then*

$$\Theta\Delta = \Theta \begin{bmatrix} U(y_1) \\ U(y_2) \\ -U(y_3) \\ -U(y_4) \end{bmatrix} = \begin{bmatrix} Q^T \\ * \\ * \\ * \end{bmatrix}.$$

Proof. For the proof, see [8, Thm. 2]. □

Theorem 4.6 provides us with a scheme for computing Q. Taking advantage of the Toeplitz structure of the matrices $U(y_i)$, we can write an algorithm for computing Q^T one row at a time that requires $12mn$ multiplications [8]. Thus combining this result with those in the previous section, Q and R^{-T} can be computed using a total of $13mn + 8n^2$ multiplications.

Recall that we only need Q^Tb, and not Q explicitly, to compute the LS solution to $Tx = b$. Using this observation, we now show how we can obtain considerable savings in computational work in computing the least squares solution. From Theorem 4.6, it follows that

$$\Theta\Delta b = \begin{bmatrix} Q^T \\ * \\ * \\ * \end{bmatrix} b = \begin{bmatrix} Q^Tb \\ * \\ * \\ * \end{bmatrix}.$$

Define the vectors $c_i \in \Re^n$ by $c_i = U(y_i)b$, and let $c \in \Re^{4n}$ be defined by $c^T = [c_1^T \; c_2^T \; c_3^T \; c_4^T]$. Then Q^Tb can be computed using the following steps.

```
For $k = 1, 2, \ldots, n-1$,
    $c := \Theta_k c$,
```

where $\Theta_k = \tilde{H}_k^{(2)}\tilde{H}_k^{(1)}\tilde{G}_k$ and b_1 is overwritten with Q^Tb.

The amount of work required to compute $c_1 = U(y_1)b$ is $mn - \frac{1}{2}n^2$ multiplications. Once c_1 is computed, c_4 can be obtained for free. Furthermore, c_2 and c_3 can be found without any computations. Each of the products $\Theta_k c$ require $12(n-k-1)$ multiplications. Thus Q^Tb can be computed with a total of $mn + \frac{11}{2}n^2$ multiplications. Combining the above method for computing Q^Tb with Algorithm BBH-L, the total work to compute the least squares solution to $Tx = b$ is $2mn + 14n^2 + O(n)$ multiplications.

In Table 4.1 we summarize the numerical complexities of the BBH, CKL, and Cybenko methods and the scheme presented in this paper. The table shows the number of multiplications required to compute the factorizations, as well as the LS solution, for each method. Cybenko's scheme is abbreviated CYB and our inverse factorization algorithm is denoted by IF. An $*$ (or $**$) in the table indicates that independent computation of that quantity is not possible by the given algorithm (or is not discussed by the particular authors).

TABLE 4.1
Numerical complexities for fast orthogonalization methods.

	BBH	CKL	CYB	Our IF
R	$mn + 6n^2$	$mn + 6n^2$	$**$	$**$
R^{-1}	$**$	$mn + 12n^2$	$*$	$mn + 8n^2$
Q	$12mn$	$12mn$	$**$	$**$
Q^Tb	$**$	$**$	$*$	$mn + \frac{11}{2}n^2$
x	$13mn + 7n^2$	$13mn + \frac{25}{2}n^2$	$9mn + 14n^2$	$2mn + 14n^2$

In practice, in addition to numerical complexities, it is important to consider space requirements. As with the above algorithms, our algorithm can be implemented using only linear storage. Specifically, the inverse QR factorization algorithm discussed in this paper can be implemented using only $O(m)$ storage as follows:

```
Initialize: $x = 0$,
For $k = 1, 2, \ldots, n-1$,
    compute $k$th row of $R^{-T} =: r^T$,
    compute $k$th entry of $Q^T b =: \sigma$,
    update solution: $x := x + \sigma * r$.
```

5. Ill-conditioned problems. In this section we consider solving ill-conditioned Toeplitz LS problems that arise in many signal and image processing applications. Specifically, we will be concerned with solving systems of the form

$$b = Tx + \eta, \tag{5.1}$$

where T is an ill-conditioned Toeplitz matrix and η is a random vector representing additive noise.

Such systems often arise from discretization of ill-posed problems in partial differential and integral equations. For example, in certain signal restoration problems, the integral equation

$$b(\tau) = \int_{-\infty}^{\infty} h(\tau - \tau_1)x(\tau_1)d\tau_1 + \eta(\tau)$$

is discretized to obtain a linear system of the form (5.1). In this case the vectors b, x, and η represent the observed signal, the original (or desired) signal, and the additive noise, respectively. The matrix T represents the degradation (or blur) in the signal.

Because of the ill-conditioned nature of T, naively solving $Tx = b$ will lead to extreme instability with respect to perturbations in b (e.g., additive noise). Stability can be attained through regularization [4], which amounts to solving the LS problem

$$\min \| \begin{bmatrix} b \\ 0 \end{bmatrix} - \begin{bmatrix} T \\ \alpha D \end{bmatrix} x(\alpha) \|_2. \tag{5.2}$$

D is a $p \times n$ matrix, called a regularization operator, and is usually chosen to be the identity matrix or some discretization of a differentiation operator [10]. Thus D is usually Toeplitz. The scalar α, called the regularization parameter, is used to control the degree of smoothness of the solution. In some applications a priori information about the signal and the degree of perturbation in b is known, and can be used to choose α [1]. But if no a priori information is known, then it may be necessary to solve (5.2) for several values of α [10].

One approach to solving regularized Toeplitz least squares problems is the PCG algorithm, which has been investigated recently by Nagy [15] and Chan, Nagy, and Plemmons [7]. It can be shown that, using an appropriate circulant preconditioner, the PCG algorithm can be an effective method for solving certain ill-conditioned Toeplitz LS problems. In this section we use a direct method, which is based on a modification of the fast orthogonalization scheme discussed in the previous sections of this paper.

Because of the remarks above on the matrix D, we will assume D is a Toeplitz matrix. Let A be the augmented matrix

$$A = \begin{bmatrix} T \\ \alpha D \end{bmatrix}.$$

Using the Toeplitz structures of T and D, partition A as

$$A = \begin{bmatrix} t_0 & u^T \\ v & T_0 \\ \alpha d_0 & \alpha f^T \\ \alpha g & \alpha D_0 \end{bmatrix} = \begin{bmatrix} T_0 & \tilde{u} \\ \tilde{v}^T & t_{m-n} \\ \alpha D_0 & \alpha \tilde{f} \\ \alpha \tilde{g}^T & \alpha d_{p-n} \end{bmatrix},$$

where u, $\tilde{v}$, f, and $\tilde{g}$ are $n-1$ vectors, v and $\tilde{u}$ are $m-1$ vectors, g and $\tilde{f}$ are $p-1$ vectors, and t_0, t_{m-n}, d_0, and d_{p-n} are scalars.

Let $A = QR$ be the QR factorization of A, where R is an $n \times n$ upper triangular matrix. Suppose R is partitioned as

$$R = \begin{bmatrix} r_{11} & z^T \\ 0 & R_b \end{bmatrix} = \begin{bmatrix} R_t & \tilde{z} \\ 0^T & r_{nn} \end{bmatrix},$$

where z and $\tilde{z}$ are $n-1$ vectors and r_{11} and r_{nn} are scalars. Then, using an approach similar to that in §2, setting $A^TA = R^TR$, we obtain the relations

$$\begin{aligned} R_b^T R_b &= R_t^T R_t + uu^T - \tilde{v}\tilde{v}^T - zz^T + \alpha^2 ff^T - \alpha^2 \tilde{g}\tilde{g}^T, \\ r_{11}^2 &= t_0^2 + v^T v + \alpha^2 d_0^2 + \alpha^2 g^T g, \end{aligned} \tag{5.3}$$

and

$$z^T = (t_0 u^T + v^T T_0 + \alpha^2 d_0 f^T + \alpha^2 g^T D_0)/r_{11}.$$

Rewriting (5.3) as

$$\begin{aligned} R_1^T R_1 &= R_t^T R_t + uu^T, \\ R_2^T R_2 &= R_1^T R_1 - \tilde{v}\tilde{v}^T, \\ R_3^T R_3 &= R_2^T R_2 - zz^T, \\ R_4^T R_4 &= R_3^T R_3 + \alpha^2 ff^T, \\ R_b^T R_b &= R_4^T R_4 - \alpha^2 \tilde{g}\tilde{g}^T, \end{aligned}$$

where R_1, R_2, R_3, and R_4 are upper triangular matrices, we can compute R one row at a time using Givens and Hyperbolic rotations. The method is analogous to that given in §2, except we are required to do an extra update and an extra downdate. Moreover, it is easy to see that R^{-1}, Q, and $Q^T b$ can easily be computed using methods similar to those given in §§3 and 4.

We remark that D is usually a banded Toeplitz matrix with small bandwidth. Thus the vector $\tilde{g}$ will have the form

$$[0 \cdots 0 \underbrace{* \cdots *}_{q}],$$

where q is small compared to n. In fact, if $D = I$, then the vectors g, f, $\tilde{g}$, and $\tilde{f}$ are all zero, and $d_0 = d_{p-n} = 1$. Thus in this case,

$$R_b^T R_b = R_t^T R_t + uu^T - \tilde{v}\tilde{v}^T - zz^T,$$

$$r_{11} = t_0^2 + v^T v + \alpha^2,$$

and

$$z^T = (t_0 u^T + v^T T_0)/r_{11}.$$

We see that for this choice of D the only change to the algorithms given in §§2, 3, and 4 is in the computation of r_{11}. Thus a regularized solution to (5.1) can be obtained in $O(mn)$ operations, and requires virtually no extra work than our nonregularized algorithm.

Also note that when D is the second difference equation,

$$D = \begin{bmatrix} 1 & & & & & & \\ -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \\ & & & & & 1 \end{bmatrix},$$

then we also have $f = \tilde{g} = 0$. In this case the only changes to the algorithms are in the computation of r_{11} and z. Again the regularized solution can be computed in only $O(mn)$ operations.

6. Numerical examples. The algorithms in this paper exploit the Toeplitz structure of the data matrix to obtain fast methods for solving Toeplitz LS problems. It is well known that the $O(n^2)$ and $O(n \log n)$ methods which solve $n \times n$ Toeplitz systems may be unstable if T is not symmetric positive definite [6], [13]. Luk and Qiao [13] analyze the fast orthogonalization scheme due to Sweet [18]. They show that if certain submatrices of T are ill conditioned then Sweet's algorithm can produce poor results.

In this section we provide some computational results to test the performance of the inverse QR factorization presented in this paper. It will be seen that our algorithm seems to perform well except when T is ill conditioned. In addition we show through numerical examples that our fast regularization scheme produces results that are comparable to those obtained by regularizing a stable $O(mn^2)$ method. Because of the ill conditioning of T, we cannot expect even a stable method to perform well when the data is noisy, which is also illustrated.

Five test matrices are used in this section, four of which are taken from Luk and Qiao [13]. These four examples are constructed so that each has a different submatrix which is ill conditioned. Two of these submatrices are T_0 and T itself, with the other two being defined as follows. T_1 is the matrix obtained by deleting the first row and last column of T, and T_2 is the leading 2×2 principal submatrix of T.

The fifth test matrix is ill conditioned and arises in signal processing applications. All examples were performed using Pro-Matlab software on an Apollo workstation. The machine epsilon for Pro-Matlab on this system is approximately 2.2204×10^{-16}.

In the first four examples we construct the matrices from [13], each with one of T, T_0, T_1, or T_2 being ill conditioned. Several experiments were performed, varying the degree of ill conditioning for each. The right-hand side b is chosen so that the exact solution is the vector e of all ones. The quantities reported will be

$$r_M = \frac{\|x_M - e\|_2}{\|e\|_2} \quad \text{and} \quad r_I = \frac{\|x_I - e\|_2}{\|e\|_2},$$

where x_M is the solution computed by Matlab (i.e., $x_M = T\backslash b$) and x_I is the solution computed using the inverse factorization scheme presented in §§3 and 4.

Example 1. Let T be defined as

$$T = \frac{1}{24}\begin{bmatrix} 8 & 4 & 2 & 1-\varepsilon \\ 4 & 8 & 4 & 2 \\ 2 & 4 & 8 & 4 \\ 1-\varepsilon & 2 & 4 & 8 \end{bmatrix}.$$

In this case T_1 is ill conditioned, with ε controlling the degree of ill conditioning, while T_0, T_2, and T are well conditioned. Table 6.1 shows r_M, r_I, and $\kappa(T_1)$ for several values of ε.

TABLE 6.1
Relative errors for Example 1.

ε	$\kappa(T_1)$	r_M	r_I
1×10^{-2}	1.7×10^{3}	2.0×10^{-16}	2.0×10^{-15}
1×10^{-3}	1.7×10^{4}	3.4×10^{-16}	1.1×10^{-15}
1×10^{-4}	1.7×10^{5}	1.1×10^{-16}	9.6×10^{-16}
1×10^{-5}	1.7×10^{6}	3.4×10^{-16}	1.3×10^{-15}
1×10^{-6}	1.7×10^{7}	1.7×10^{-16}	1.3×10^{-15}
1×10^{-7}	1.7×10^{8}	1.1×10^{-16}	1.3×10^{-15}

Example 2. In this example T is defined as

$$T = \begin{bmatrix} 1 & 1-\varepsilon & 2 & 0 & 0 & 0 \\ 1-\varepsilon & 1 & 1-\varepsilon & 2 & 0 & 0 \\ 2 & 1-\varepsilon & 1 & 1-\varepsilon & 2 & 0 \\ 0 & 2 & 1-\varepsilon & 1 & 1-\varepsilon & 2 \\ 0 & 0 & 2 & 1-\varepsilon & 1 & 1-\varepsilon \\ 0 & 0 & 0 & 2 & 1-\varepsilon & 1 \end{bmatrix},$$

where in this case T_2 is ill conditioned and T_0, T_1, and T are well conditioned. Table 6.2 shows the results for this example.

TABLE 6.2
Relative errors for Example 2.

ε	$\kappa(T_2)$	r_M	r_I
1×10^{-2}	2.0×10^{2}	3.9×10^{-16}	4.6×10^{-14}
1×10^{-3}	2.0×10^{3}	4.6×10^{-16}	3.9×10^{-14}
1×10^{-4}	2.0×10^{4}	1.2×10^{-15}	1.0×10^{-13}
1×10^{-5}	2.0×10^{5}	4.3×10^{-16}	4.3×10^{-14}
1×10^{-6}	2.0×10^{6}	1.2×10^{-15}	1.7×10^{-13}
1×10^{-7}	2.0×10^{7}	6.0×10^{-16}	1.6×10^{-14}

Example 3. In this example we let

$$T = \frac{1}{7}\begin{bmatrix} 1 & 3 & 1-\varepsilon & -1 \\ 3 & 1 & 3 & 1-\varepsilon \\ 1-\varepsilon & 3 & 1 & 3 \\ -1 & 1-\varepsilon & 3 & 1 \end{bmatrix},$$

where, in this case, T_1, T_2, and T are well conditioned and T_0 is ill conditioned. The results for this example are shown in Table 6.3.

TABLE 6.3
Relative errors for Example 3.

ε	$\kappa(T_0)$	r_M	r_I
1×10^{-2}	5.8×10^{2}	1.9×10^{-16}	4.5×10^{-16}
1×10^{-3}	5.8×10^{3}	3.2×10^{-16}	1.1×10^{-15}
1×10^{-4}	5.8×10^{4}	2.3×10^{-16}	1.1×10^{-15}
1×10^{-5}	5.8×10^{5}	2.0×10^{-16}	3.2×10^{-15}
1×10^{-6}	5.8×10^{6}	2.3×10^{-16}	8.0×10^{-16}
1×10^{-7}	5.8×10^{7}	2.2×10^{-16}	7.6×10^{-16}

Example 4. In this example we consider the ill-conditioned matrix

$$T = \frac{1}{27}\begin{bmatrix} 27 & 9 & 3 & -23+\varepsilon \\ 9 & 27 & 9 & 3 \\ 3 & 9 & 27 & 9 \\ -23+\varepsilon & 3 & 9 & 27 \end{bmatrix}.$$

Table 6.4 shows the results for this example, where we see that the inverse QR algorithm performs poorly due to the ill conditioning of T. We note, though, that in most practical examples we can expect that the right-hand side, b, is not known exactly. Instead, what is known is $b+\eta$, where η represents additive noise. In this case, as discussed in §5, regularization is often used to compute a meaningful solution. In the next example, we illustrate that the inverse factorization algorithm can perform well for these ill-conditioned problems.

Example 5. In this example we consider solving an ill-conditioned problem of the form (5.1). Let T be the 100×100 Toeplitz matrix whose (i, j) entry is given by

$$t_{ij} = \begin{cases} 0 & \text{if } |i-j| > 8, \\ \frac{4}{51} g(0.15, x_i - x_j) & \text{otherwise}, \end{cases}$$

TABLE 6.4
Relative errors for Example 4.

ε	$\kappa(T)$	r_M	r_I
1×10^{-2}	5.7×10^{3}	1.0×10^{-13}	3.9×10^{-9}
1×10^{-3}	5.7×10^{4}	4.4×10^{-12}	2.3×10^{-7}
1×10^{-4}	5.7×10^{5}	3.9×10^{-12}	9.6×10^{-6}
1×10^{-5}	5.7×10^{6}	3.9×10^{-11}	3.0×10^{-4}
1×10^{-6}	5.7×10^{7}	4.4×10^{-10}	2.3×10^{-1}
1×10^{-7}	5.7×10^{8}	9.3×10^{-9}	2.3×10^{-1}

where

$$x_i = \frac{4i}{51}, \qquad i = 1, 2, \ldots, 100$$

and

$$g(\sigma, \gamma) = \frac{1}{2\sqrt{\pi}\sigma} \exp\left(-\frac{\gamma^2}{4\sigma^2}\right).$$

Matrices of this form occur in signal restoration applications, and are used to model certain degradations in the recorded signal [3], [10], [12]. The spectral condition number of T is approximately 2.43×10^6.

To simulate a signal restoration application, we first constructed the (original) signal x with 100 data points as shown in Fig. 6.1. We then constructed the degraded, noisy (i.e., observed) signal as shown in Fig. 6.2, by computing

$$b = Tx + \eta,$$

where η is a vector whose elements were chosen randomly from a uniform distribution in $(0, 10^{-3})$. For this example $\|\eta\|_2 \approx 5.9 \times 10^{-3}$.

Our goal, given T and b, is to recover an approximation to the original signal x. In Fig. 6.3, we show the computed solution using our inverse factorization algorithm. The solution computed using Matlab (i.e., $x = T\backslash b$) also produces poor results, the graph of which looks very similar to Fig. 6.3. In fact, if we denote the solutions computed by Matlab and the inverse factorization algorithm by x_M and x_I, respectively, then our computed solutions satisfied $\|x_M - x_I\|_2 / \|x_M\|_2 \approx 1.2 \times 10^{-5}$. We see that, due to the ill-conditioned nature of T and the presence of additive noise, neither method performs well.

To obtain a reasonable solution, we use the method of regularization as discussed in §5. Here we take $D = I$ and $\alpha = 2.1 \times 10^{-4} \approx \|\eta\|_2 / \|x\|_2$. In Fig. 6.4, we show the computed solution, $\hat{x}_I$, to (5.2) using the fast regularized inverse QR method discussed in §5. The result computed by Matlab, $\hat{x}_M = \hat{T}\backslash\hat{b}$ where

$$\hat{T} = \begin{bmatrix} T \\ \alpha I \end{bmatrix} \quad \text{and} \quad \hat{b} = \begin{bmatrix} b \\ 0 \end{bmatrix},$$

produced similar results. In this case $\|x_M - x_I\|_2 / \|x_M\|_2 \approx 7.3 \times 10^{-10}$. Thus, both methods produced solutions with similar accuracy.

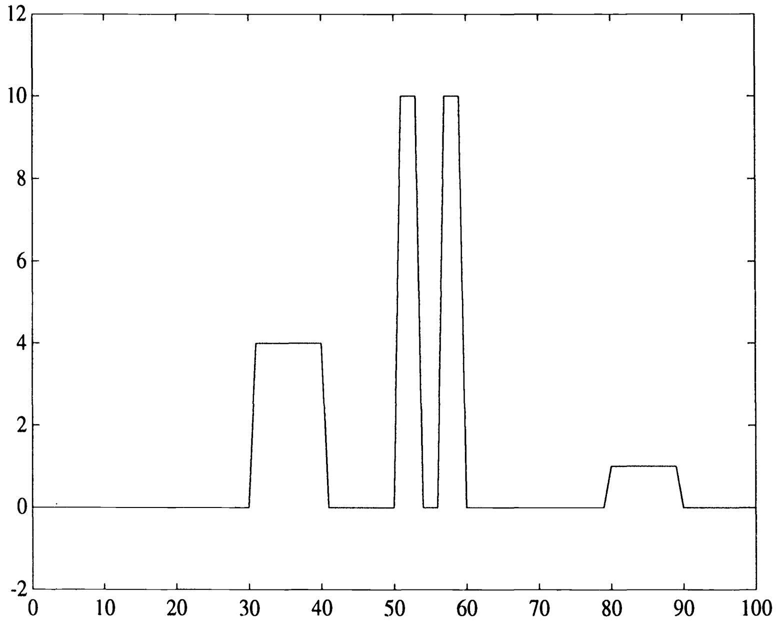


FIG. 6.1. *Original signal.*

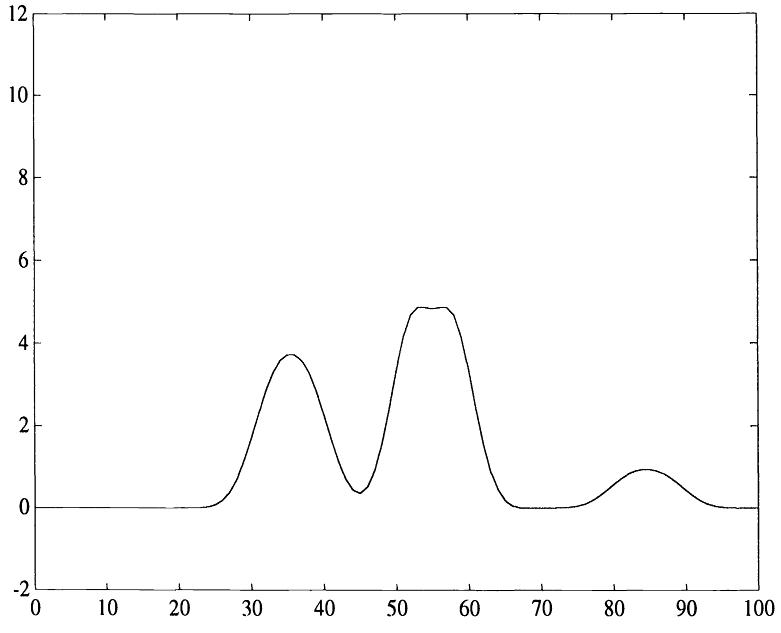


FIG. 6.2. *Blurred, noisy signal.*

7. Concluding remarks. In this paper we described the fast orthogonalization schemes of Bojanczyk, Brent, and de Hoog [5] and Chun, Kailath, and Lev-Ari [8]. We have extended these results to obtain an $O(mn)$ method for computing the inverse QR factorization of a Toeplitz matrix. Although our numerical results have shown that our

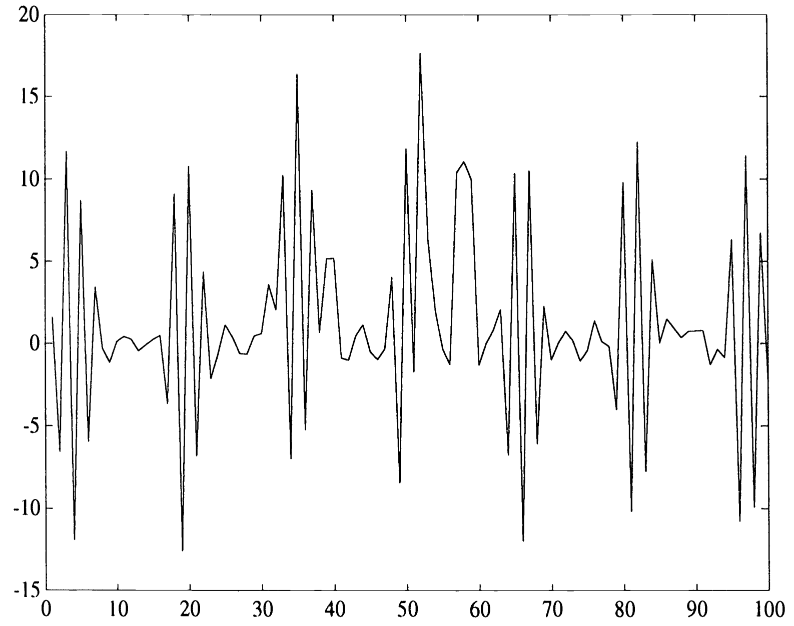


FIG. 6.3. *Computed solution using our inverse scheme with no regularization.*

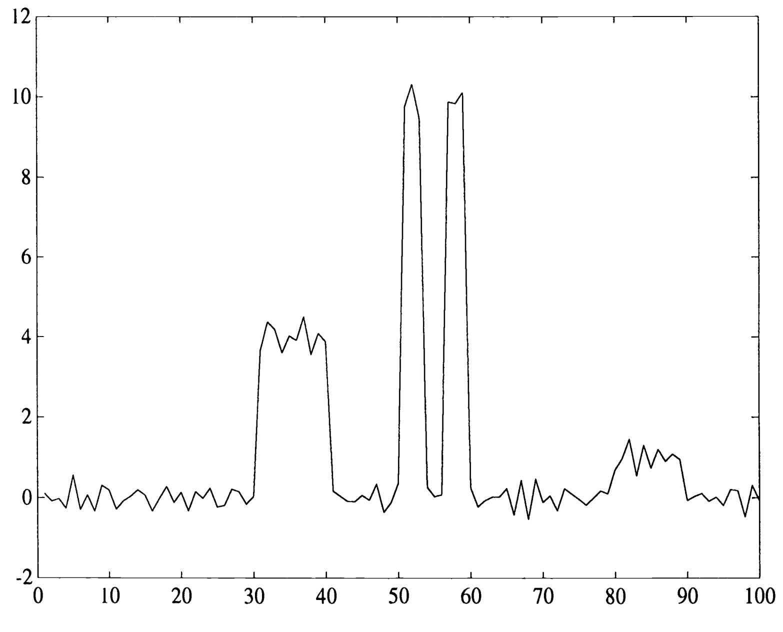


FIG. 6.4. *Computed solution using regularized inverse scheme.*

algorithm may not perform well when T is ill conditioned, we showed how our algorithm can be easily modified to incorporate regularization. A numerical example was presented showing that our regularized algorithm may be useful in solving certain ill-conditioned signal restoration problems.

Finally, we remark that further computational savings can be attained. Specifically, we can use fast Fourier transforms (FFTs) to compute the Toeplitz matrix-vector product $T_0^T v$, required in the initialization stage of Algorithm BBH-L. Using FFTs, this can be done in $O(m \log n)$ operations, rather than $O(mn)$. Furthermore, our numerical results indicate that, for at least some ill-conditioned problems, the error in the computed solution from the inverse QR scheme behaves like $\kappa^2(T)$. Thus, we may consider using the seminormal equations $R^T R x = T^T b$, since this would reduce the cost in computing x to $2mn + 9n^2$ multiplications (or less, if FFTs are used). In particular, the solution $x = R^{-1} R^{-T} T^T b$ can be computed using the BBH-L approach (or the regularized BBH-L approach from §5), which can also be implemented using only linear storage.

REFERENCES

[1] J. B. Abbiss and P. G. Earwicker, *Compact operator equations, regularization and super-resolution*, in Mathematics in Signal Processing, Clarendon Press, Oxford, 1987, pp. 225–236.

[2] S. T. Alexander, C.-T. Pan, and R. J. Plemmons, *Analysis of a recursive least squares hyperbolic rotation algorithm for signal processing*, Linear Algebra Appl., 98 (1988), pp. 3–40.

[3] H. Andrews and B. Hunt, *Digital Image Restoration*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[4] Å. Björck, *Least squares methods*, in Handbook of Numerical Methods, Vol. 1, P. Ciarlet and J. Lions, eds., Elsevier, North-Holland, Amsterdam, 1989.

[5] A. W. Bojanczyk, R. P. Brent, and F. R. de Hoog, *QR factorization of Toeplitz matrices*, Numer. Math., 49 (1986), pp. 81–94.

[6] J. R. Bunch, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 349–364.

[7] R. H. Chan, J. G. Nagy, and R. J. Plemmons, *Circulant preconditioned Toeplitz least squares iterations*, SIAM J. Matrix Anal. Appl., to appear.

[8] J. Chun, T. Kailath, and H. Lev-Ari, *Fast parallel algorithms for QR and triangular factorization*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 899–913.

[9] G. Cybenko, *Fast Toeplitz orthogonalization using inner products*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 734–740.

[10] L. Eldén, *An algorithm for the regularization of ill-conditioned, banded least squares problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 237–254.

[11] G. H. Golub and C. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.

[12] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[13] F. T. Luk and S. Qiao, *A fast but unstable orthogonal triangularization technique for Toeplitz matrices*, Linear Algebra Appl., 88/89 (1987), pp. 495–506.

[14] D. G. Manolakis and J. G. Proakis, *Introduction to Digital Signal Processing*, Macmillan, New York, 1988.

[15] J. G. Nagy, *Toeplitz least squares computations*, Ph.D. thesis, North Carolina State University, Raleigh, NC, 1991.

[16] J. G. Nagy and R. J. Plemmons, *Some fast Toeplitz least squares algorithms*, in Conf. Advanced Signal Processing Algorithms, Architectures, and Implementations II, Vol. 1566, San Diego, CA, 1991.

[17] C.-T. Pan and R. J. Plemmons, *Least squares modifications with inverse factorization: parallel implications*, Comput. Appl. Math., 27 (1989), pp. 109–127.

[18] D. R. Sweet, *Fast Toeplitz orthogonalization*, Numer. Math., 43 (1984), pp. 1–21.

[19] ———, *Fast block Toeplitz orthogonalization*, Numer. Math., 58 (1991), pp. 613–629.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1194–1213, September 1993

USING IMPLICIT ODE METHODS WITH ITERATIVE LINEAR EQUATION SOLVERS IN SPECTRAL METHODS*

IVAR LIE[†]

Abstract. The use of Chebyshev spectral methods in space on an evolution partial differential equation results, in many cases, in a stiff system of ordinary differential equations (ODEs). ODE solvers based on explicit methods will therefore be inefficient for such equations. The use of conventional implicit ODE solvers is difficult since the Jacobian matrix of the ODE system is full and large.

It is shown how to apply ODE solvers with iterative linear equation solvers to ODEs coming from spectral discretizations. These ODE solvers do not use the Jacobian explicitly, but good preconditioners for the Newton matrix are needed for their efficient operation. Preconditioners of the tensor product type are developed for certain classes of hyperbolic partial differential equations (PDEs), and numerical experiments show that these preconditioners give a good performance of the ODE solver and are a substantial improvement over the performance of explicit solvers.

Key words. partial differential equations, spectral approximations, preconditioning, ordinary differential equations

AMS subject classifications. 65N35, 65F10

1. Introduction.

Consider the numerical solution of the evolution equation

$$u_t = Lu + f \quad \text{on } \Omega \times [0,T], \quad \Omega \subset \mathbb{R}^d \tag{1.1}$$

with appropriate initial and boundary conditions. L denotes a differential operator in space, and f and u are assumed to be sufficiently regular.

When we discretize (1.1) in space by spectral methods using N_i points in each dimension, we get a set of ordinary differential equations (ODEs):

$$V_t = F(x,t,V), \quad V \in \mathbb{R}^N, \quad F : \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N \tag{1.2}$$

where $N = \prod_{i=1}^d N_i$.

Using Chebyshev spectral methods in one space dimension, it is well known that the spectral radius of the differentiation matrix is $\Theta(N^2)$ and that there are eigenvalues $\Theta(1)$, see, e.g., [25]. (We say that $f(N) = \Theta(g(N))$ if there exist positive constants c and C such that $c\,g(N) \le f(N) \le C\,g(N)$.) Hence the resulting ODE system will be stiff for moderate and large values of N, even for linear partial differential equations (PDEs).

An implicit ODE solver should therefore be more efficient in solving (1.2) than an explicit solver, and that is illustrated in the following for a model problem. Stability considerations based on stability regions and eigenvalues for the Jacobian matrix of the ODE system are not straightforward for spectral methods. However, by using so-called pseudo eigenvalues instead of the real eigenvalues, the standard theory still applies; for a discussion of this, see, e.g., [21].

The differentiation matrices for spectral methods are full, and that implies that the Jacobian for (1.2) is full even for the simplest linear PDE. It is evident that for two-dimensional and three-dimensional problems and moderate values of N the Jacobians will be intractable both in terms of storage and computation. Hence we will have to use solvers that do not use the Jacobian explicitly. So-called reduced storage iterative methods will be described briefly in §2. These methods require good preconditioners to the Newton matrix, and the main part of this paper contains development of such

*Received by the editors November 8, 1989; accepted for publication (in revised form) October 8, 1992.
[†]NDRE Division for Electronics, P.O. Box 25, N-2007 Kjeller, Norway (ivar@ffi.ndre.no).

preconditioners. For multidimensional problems it turns out that the tensor product-type of preconditioners works well. The use of tensor product-type of formulas in the numerical solution of PDEs is well known, see, e.g., [18]. In the development of the preconditioners we use a preconditioning matrix derived by Funaro [8] for the Chebyshev spectral differentiation matrix. The reason for not using the differentiation matrix itself is that it is difficult to invert and that the preconditioning matrix is a product of two matrices that are easily computed.

A variable-stepsize implicit ODE solver will, in many cases, be more efficient than a fixed-stepsize implicit method (Crank–Nicholson or backward Euler, for example), since the optimal stepsize for a prescribed error tolerance will vary considerably during the integration interval.

Fixed-step implicit integration and semi-implicit integration are extensively used in spectral methods, see, e.g., [19] for the use of the Adams–Moulton family of methods, [17] for a discussion of the so-called Lerat schemes, and [7] for a discussion of the Morchiosne scheme, which can be constructed to have arbitrary high accuracy.

In the following, we will consider the use of implicit ODE solvers. Looking at the eigenvalues of the Chebyshev differentiation matrix D, see, e.g., [25], we see that there are eigenvalues close to the imaginary axis $\Theta(N)$ from the origin. Hence it is likely that an A-stable method is a good choice for integration.

The purpose of the two tables below is to show that an implicit solver really takes far fewer steps than an explicit solver for the one-dimensional model problem:

$$u_t + Au_x = 0 \quad \text{on } [-1,1] \tag{1.3}$$

where

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad A = \begin{bmatrix} a & 0 \\ -1 & b \end{bmatrix}$$

with $a > 0, b > 0$ and the boundary condition: $u(-1,t) = 0$. No boundary condition is specified at $x = 1$. The initial function is $\exp(-10x^2)$. In the experiment we have set $a = 1$, the time integration interval is $[0.0, 2.0]$ and the error tolerance is 10^{-4}.

We use the implicit ODE solver SIMPLE [24], which is based on a singly diagonally implicit Runge–Kutta pair. By choosing $b \gg a$, for example, we can model two interacting waves, one fast and one slow. The analytical solution of (1.3) is used to compute the actual error in the computation.

The implicit solver is also more efficient even if each step costs more as shown in Tables 1.1 and 1.2 below. The CPU times are given in seconds. The explicit solver DOPRI5 is described in [13].

TABLE 1.1
Number of steps for model problem using SIMPLE.

N	17	17	17	33	33
b	10	20	100	10	20
no. of steps	37	32	33	36	36
CPU time	5.2	4.4	4.6	5.0	5.0

We see that SIMPLE (and hopefully other implicit solvers) has the nice property that the number of steps is almost independent of the values of N and b, whereas for DOPRI5 (and other explicit solvers) the number of steps is $\Theta(N^2 b)$. The two methods produced solutions with almost the same global error.

TABLE 1.2
Number of steps for model problem using DOPRI5.

N	17	17	17	33	33
b	10	20	100	10	20
no. of steps	186	351	889	669	1291
CPU time	9.8	19.2	48.0	36.2	64.8

2. Implicit ODE methods with iterative linear solvers. In this section we will describe briefly ODE solvers where we don't have to compute the Jacobian explicitly.

All implicit ODE solvers for the generic ODE $y'(t) = f(t, y)$ will have to solve one or several systems of nonlinear algebraic equations of the form

$$y = \gamma + h\beta f(t, y), \qquad \beta \in \mathbb{R}, \gamma \in \mathbb{R}^n \tag{2.1}$$

for each step.

Most ODE solvers use a variant of modified Newton iteration to solve these equations, and the essence of such an iteration is the solution of a linear system

$$B \cdot (y^{[i+1]} - y^{[i]}) = -(y^{[i]} - \gamma - h\beta f(t, y^{[i]})) \tag{2.2}$$

where $B = I - h\beta J$, $y^{[i]}$ denotes iterate i of y, and J is some approximation to $\partial f/\partial y|_{y=y^{[i]}}$. The usual method is to use an LU factorization of B adapted to the sparsity structure and corresponding forward and backward substitutions. B can, in many cases, be used over several steps which reduces the cost of the iteration.

The main problem in spectral methods is, as remarked above, that J is full and it will be prohibitively costly to store and operate with it for large ODE systems. An alternative way to proceed is to solve (2.2) iteratively. In many iterative methods B is not needed explicitly; what is needed is $B \cdot v$, where v is a vector of appropriate dimension. Hence, we avoid storing and factoring of B. Two of the most efficient classes of iterative methods in use today are the Krylov subspace based methods, e.g., the conjugate gradient method for symmetric positive definite systems and IOM and GMRES for nonsymmetric systems. See [3] and [2] for a description of these methods in the ODE context. Krylov subspace methods can also be combined with globally convergent methods for the solution of nonlinear equations; see [4]. Other iterative techniques for ODEs are discussed in [11], [6], and the recent survey [22].

In [2] a so-called matrix-free iterative method is described. It is based on approximating $B \cdot v$ by the difference quotient

$$v - h\beta \frac{f(y + \sigma v) - f(y)}{\sigma}, \qquad \sigma \in \mathbb{R}.$$

This method is not very efficient unless the spectrum of J is tightly clustered around a few eigenvalues (see [2]), and that is definitely not the case for most semidiscretizations of PDEs. In addition, we have to consider the error contribution from the difference approximation to $B \cdot v$ and the coupling between the local error and the error tolerance for the iterative solution of (2.2).

Some form of preconditioning of B is therefore needed to speed up the iterative methods. In [3] the so-called reduced-storage methods are discussed and analyzed. For these methods the linear system

$$Bx = b$$

is replaced by

$$\hat{S}^{-1} P_1^{-1} B x = \hat{S}^{-1} P_1^{-1} b \tag{2.3}$$

where $\hat{S}$ is a diagonal scaling matrix and P_1 is the preconditioning matrix. It is also possible to use right preconditioning and scaling, but we will not use these in this paper. The requirements for a good preconditioner are as usual: P_1 should be a good approximation to B, and linear systems $P_1 x = b$ should be easy to solve compared to solving $Bx = b$. When using iterative methods for solving (2.3) we need to compute $\hat{S}^{-1} P_1^{-1} B v$, and that can often be done without storing B and P_1.

The work reported in [3] resulted in the code LSODPK, based on the LSODE family of ODE solvers. The reduced-storage methods were tested on several relatively large test problems in [3], and the results are encouraging for IOM, GMRES, and appropriate preconditioners. The choice of preconditioners for certain classes of problems is also discussed in [3].

3. A hyperbolic system in one space dimension. Now consider the Chebyshev spectral discretization of the one-dimensional model equation (1.3) on N points. We get the following set of ODEs:

$$V_t + A \otimes D \cdot V = 0 \tag{3.1}$$

where $V \in \mathbb{R}^{2N}$, and we see that the Jacobian is $J = -A \otimes D$, and hence we have $\rho(J) = \Theta(N^2 b)$ by the properties of D (see, e.g., [25]), and the properties of the Kronecker product of matrices, see, e.g., [16].

Funaro [8] has constructed an efficient preconditioner, Q, for the differentiation matrix and has shown that $1 \le \lambda(Q^{-1}D) < \frac{\pi}{2}$. His result is as follows.

LEMMA 3.1. *The preconditioning matrix Q for the Chebyshev differentiation matrix is given by*: $Q = Z\tilde{D}$. *$\tilde{D}$ is the upwind difference operator for the grid points $\{x_j\}_{j=0}^{N-1}$, $x_j = \cos\frac{j}{N}\pi$, and Z is the matrix of the interpolation operator $\mathcal{Z} : \mathbb{R}^N \to \mathbb{R}^N$ from the points $\xi_j = \cos\frac{2j-1}{2N}\pi$ to the points $x_j = \cos\frac{j}{N}\pi$. For a polynomial of degree $N-1$:*

$$\{\phi'(\xi_1), \dots, \phi'(\xi_N)\} \overset{\mathcal{Z}}{\mapsto} \{\phi'(x_1), \dots, \phi'(x_N)\}.$$

The elements of Z and $\tilde{D}$ are the following:

$$\tilde{d}_{ij} = \begin{cases} -1/(x_{i-1} - x_i) & j = i \\ 1/(x_{i-1} - x_i) & j = i+1 \\ 0 & \textit{otherwise} \end{cases}$$

$$z_{ik} = \frac{(-1)^{i+k-1}\sqrt{1-\xi_k^2}}{(x_i - \xi_k)N}.$$

Proof. The proof is $\tilde{D}$: straightforward, since $\tilde{D}$ is the upwind difference operator for $\{x_j\}_{j=0}^{N-1}$. Consider the Lagrange interpolation polynomials $\{\ell_j\}$ on the points $\{\xi_i\}$. We then have:

$$\{\ell_j'(x_i)\} = Z \cdot \{\ell_j'(\xi_i)\}$$

or

$$Z = \{\ell_j'(x_i)\} \cdot \{\ell_j'(\xi_i)\}^{-1}.$$

Now apply the interpolation formula:

$$\ell_j'(x) = \sum_{k=1}^{N} \ell_j'(\xi_k)\,\ell_k(x) = \sum_{k=1}^{N} \ell_j'(\xi_k)\frac{T_N(x)}{T_N'(\xi_k)(x-\xi_k)}.$$

We have $T_N(x_i) = (-1)^i$ and $T_N'(\xi_k) = N\sin\frac{(2k-1)}{2}\pi/\sqrt{1-\xi_k^2}$ and

$$\ell_j'(x_i) = \sum_{k=1}^{N} \ell_j'(\xi_k)\frac{(-1)^{i+k+1}\sqrt{1-\xi_k^2}}{N(x_i-\xi_k)} = \sum_{k=1}^{N} \ell_j'(\xi_k) z_{ik}$$

and the result follows. □

Note that the $\tilde{D}$ matrix in [8] is different from ours. The operator in [8] is given for the grid $\{x_j\}_{j=1}^{N}$ and zero boundary conditions to the right, even if it is not explicitly stated in the paper. An explicit expression for the elements in the preconditioned differentiation matrix $Q^{-1} \cdot D$ can also be found in [8], and this expression will also be slightly different in our case.

Since

$$\lambda((A^{-1} \otimes Q^{-1})(A \otimes D)) = \lambda(I_2 \otimes Q^{-1}D) = \lambda(Q^{-1}D)$$

we see from the results of [8] that $-A \otimes Q$ will be a good preconditioner for the Jacobian J, and for the Newton matrix $I + h\beta A \otimes D$ we may use the preconditioner $I + h\beta A \otimes Q$. The advantages of using Q instead of D in the preconditioner for the Newton matrix are not so evident, since it is not known how to factor and solve linear systems using $I + \bar{h}\beta A \otimes Q$ significantly faster than using $I + \bar{h}\beta A \otimes D$. One advantage of using Q is that the D matrix is difficult to compute with high precision and D^{-1} may not be very accurate. The matrix Q is easier to compute.

The preconditioners to be presented in the following three sections will be derived using the D matrix. The matrix Q can be substituted for D in all these preconditioners.

In ODE solvers with iterative linear solvers, such as LSODPK [3] and KRYSI [14], linear systems with the preconditioner are solved for each inner iteration and for every stage, and the preconditioner is used (with an old value of h) as long as the iteration converges at a reasonable rate. When convergence problems occur, the preconditioner is updated with the current value of h and, if needed, a reevaluation of the Jacobian approximation is performed. If the linear system with the preconditioner is solved by direct methods, the new Newton matrix is factored to save time in the solution process.

Large variations in stepsize during an integration will imply more reevaluation and refactorizations than for slowly varying stepsize. In order to study the effect of these variations, let

$$\lambda_{ij} = \lambda((I + \bar{h}\beta A \otimes D)^{-1}(I + h\beta A \otimes D))$$

where $\bar{h}$ is the stepsize used in the last factorization. From standard theorems in matrix theory (see [16]), we have:

$$\lambda_{ij} = \frac{1 + h\beta\lambda_{A,i}\lambda_{D,j}}{1 + \bar{h}\beta\lambda_{A,i}\lambda_{D,j}}$$

where $\lambda_{A,i}$ and $\lambda_{D,j}$ are the eigenvalues of A and D, respectively. Let $h = \kappa\bar{h}$ and $\beta\lambda_{A,i}\lambda_{D,j} = \gamma_{ij}$. Then

$$\lambda_{ij} = \frac{1+\kappa\bar{h}\gamma_{ij}}{1+\bar{h}\gamma_{ij}} = 1 + \frac{\bar{h}\gamma_{ij}(\kappa - 1)}{1+\bar{h}\gamma_{ij}}.$$

Let $z = \bar{h}\gamma_{ij}$ and hence we have

$$|\lambda_{ij}| \leq \frac{1+\kappa|z|}{|1+z|},$$

and if $\kappa < 1$, then λ_{ij} is contained in a small region independent of the size of $|z|$. If κ is large, λ_{ij} is contained in a larger region and the convergence of the modified Newton method will be bad. As an example of how the above mapping works, consider the region $-\frac{1}{2} \leq \Re z \leq 0$, $|\Im z| \leq 10$, which is supposed to contain the spectrum of a scaled differentiation matrix. For $\kappa = \frac{1}{2}$ this region is mapped into the circle $|\lambda - 1| = \frac{1}{2}$, but with the circle $|\lambda - \frac{3}{4}| = \frac{1}{4}$ cut out of it. For even smaller κ the circles become smaller, but the center of the largest one is always $(1,0)$. For $\kappa = 4$ the region is mapped into the circle $|\lambda - 1| = 3$, but with the circle $|\lambda - 2.5| = 1.5$ cut out of it. This means that for $\kappa < 1$, i.e., the stepsize has decreased since the last factorization, the linear iteration should converge quickly. For increasing stepsize ($\kappa > 1$) the eigenvalues are more widely spread and the linear iteration may converge more slowly.

Using Q instead of D in the preconditioner should not alter these results significantly since the spectra of Q and D are close.

In order to show that an ODE solver with iterative linear solver works reasonably well for our test example, we have run the same experiment whose results are shown in Tables 1.1 and 1.2, on the ODE solver KRYSI [14] where the linear system for the preconditioner is solved by LU factorization. The matrix Q is used in the construction of the preconditioner. We see from Table 3.1 that the number of steps is practically independent of the stiffness of the problem. SIMPLE uses fewer steps than KRYSI, but that is fairly common, at least for small problems; see the numerical tests in [14] and [3].

TABLE 3.1
Number of steps for model problem using KRYSI.

N	17	17	17	33	33
b	10	20	100	10	100
no. of steps	51	49	41	51	40

For our one-dimensional model problem and other one-dimensional problems we cannot save much space for J by using iterative methods because the matrix D is needed, anyway, and the number of PDEs is assumed to be small compared to N. The savings will come on two-dimensional and three-dimensional problems where the Jacobian is so large that using implicit ODE solvers in the conventional way will be inefficient. In addition we will see that a tensor product type of preconditioner works well for two-dimensional problems.

4. A hyperbolic PDE in two space dimensions. Now consider the following model problem in two space dimensions:

$$v_t + c_1 v_x + c_2 v_y = 0, \qquad c_1, c_2 \in \mathbb{R} \tag{4.1}$$

by Chebyshev spectral methods with M points in the x-direction and N points in the y-direction.

The resulting ODE system can be written as follows:

$$V_t + c_1(I_N \otimes D_1)V + c_2(D_2 \otimes I_M)V = 0 \tag{4.2}$$

where $V \in \mathbb{R}^{NM}$ and D_1 and D_2 are the differentiation matrices in the x- and y-directions, respectively. Note that $N = M \Rightarrow D_1 = D_2$.

The Jacobian is:

$$J = -c_1(I_N \otimes D_1) - c_2(D_2 \otimes I_M) \in \mathbb{R}^{NM \times NM}$$

and from [16] we have the following:

$$\lambda_{ij}(J) = -c_1\lambda_i(D_1) - c_2\lambda_j(D_2)$$

$$\rho(J) = c_1\rho(D_1) + c_2\rho(D_2).$$

We see that $-J$ is the Kronecker sum of the matrices c_1D_1 and c_2D_2.

If $M = N$, then $\rho(J) = (c_1 + c_2)\rho(D_1)$, so the spectral radius for J in the two-dimensional case is not significantly larger than for an equivalent one-dimensional problem if c_1 and c_2 are of the same order of magnitude.

The Newton matrix is $B = I_{MN} + h\beta(c_1(I_N \otimes D_1) + c_2(D_2 \otimes I_M))$. We may construct a preconditioner P_2 for B with the same structure as B. The only difference between B and P_2 is that in the latter the stepsize used at the time of preconditioner evaluation is applied instead of the current stepsize. In the following the current stepsize is denoted by h and the stepsize in the preconditioner is denoted by $\hat{h}$. The solution of linear systems with P_2 is obviously as costly as using the matrix B itself.

If we use a linear solver of Krylov subspace type, we need to compute Bv, $v \in \mathbb{R}^{NM}$ (if this is not done by differences as discussed in §2). The terms of Bv can be computed as follows: If we partition the vector v into N blocks v_i of size M, the product $(I_N \otimes D_1)v$ is:

$$\begin{bmatrix} D_1 & & & \\ & D_1 & & \\ & & \ddots & \\ & & & D_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}.$$

This computation can be performed by doing N matrix-vector multiplications of size M. The cost is $\Theta(M^2N)$ and only D_1 and v need to be stored.

Alternatively, if we let $\{v_i\}$ form the columns of a matrix $\mathcal{V}$, i.e., $v = \operatorname{vec}(\mathcal{V})$ in the notation of [16, §12.1], we can use the relation

$$(I_N \otimes D_1)\ \operatorname{vec}(\mathcal{V}) = \operatorname{vec}(D_1\mathcal{V}) \tag{4.3}$$

and compute the required matrix-vector product by doing the matrix multiplication $D_1\mathcal{V}$ and transform back to a vector if needed. The cost of this operation is of course the same as for the former method.

For the term $(D_2 \otimes I_M)v$ we use the same partitioning of the vector v and we can write:

$$\begin{bmatrix} d_{11}I_M & d_{12}I_M & \cdots & d_{1N}I_M \\ d_{21}I_M & d_{22}I_M & \cdots & d_{2N}I_M \\ \vdots & \vdots & \ddots & \vdots \\ d_{N1}I_M & d_{N2}I_M & \cdots & d_{NN}I_M \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}$$

where $D_2 = \{d_{ij}\}$. We see that one block of the resulting vector is obtained by N scalar-vector multiplications, i.e., $\Theta(N^2)$ operations. The complete vector is then computed in $\Theta(MN^2)$ operations, and only storage for D_2 and v is needed. Note that we don't have to form the two large block matrices explicitly. An alternative computation here is based on the relation

$$(D_2 \otimes I_M)\text{vec}(\mathcal{V}) = \text{vec}(\mathcal{V}D_2^T), \tag{4.4}$$

and we see that we can form the matrix-vector product above by computing $\mathcal{V}D_2^T$.

The total cost is $\Theta(M^2N + N^2M)$. We could have performed the Chebyshev differencing by fast Fourier transforms (FFTs) using $\Theta(NM(\log_2 N + \log_2 M))$ operations. The fastest alternative in practice will depend on N and M as well as computer architecture and implementation aspects, see, e.g., [23]. As a rule of thumb, matrix multiplication is fastest for $M, N \leq 100$.

We can write the equation $Bx = b$ in the following way using the relations (4.3), (4.4), and the result in [16, §12.1, Cor. 2].

LEMMA 4.1. *If* $A_1 \in \mathbb{R}^{m\times m}$, $A_2 \in \mathbb{R}^{n\times n}$, *and* $X \in \mathbb{R}^{m\times n}$, *we have*:

$$\text{vec}(A_1X + XA_2) = (I_n \otimes A_1 + A_2^T \otimes I_m)\ \text{vec}(X).$$

Applying the lemma to $Bx = b$ gives immediately:

$$(I + \hat{h}\beta c_1 D_1)X + \hat{h}\beta c_2 X D_2^T = \mathcal{B}, \qquad b = \text{vec}(\mathcal{B}). \tag{4.5}$$

This equation has a unique solution if the matrices $I + \hat{h}\beta c_1 D_1$ and $-\hat{h}\beta c_2 D_2$ have no eigenvalues in common; see [16, Thm. 2, p. 414]. This holds, e.g., when $c_1 > 0$, $c_2 > 0$. We can find the solution of (4.5) by using a result in [1, Chap. 10, Misc. Ex. 21].

LEMMA 4.2. *Consider the matrix equation* $X = C + \epsilon(AX + XB)$, $\epsilon \in \mathbb{R}$ *and assume that* $|\epsilon|(\|A\| + \|B\|) < 1$. *The solution is given by*

$$X = C + \sum_{n=1}^{\infty} \epsilon^n \Phi_n(A, B) \tag{4.6}$$

where Φ_n *is given by the recurrence*:

$$\Phi_n = A\Phi_{n-1} + \Phi_{n-1}B, \qquad \Phi_0 = C$$

or by the explicit formula

$$\Phi_n = \sum_{i=0}^{n} A^{n-i}CB^i \binom{n}{i}.$$

In our case we can write the equation as follows:

$$X - \hat{h}\beta(c_1 D_1 X + c_2 X D_2^T) = \mathcal{B}$$

and application of the above lemma gives directly:

$$X = \mathcal{B} + \sum_{n=1}^{\infty} (\hat{h}\beta)^n \sum_{i=0}^{n} (c_1 D_1)^{n-i} \mathcal{B} (c_2 D_2^T)^i \binom{n}{i}. \tag{4.7}$$

From this expression we can construct polynomial preconditioners recursively. Note that the convergence requirement for the series in the lemma implies a restriction on the stepsize: $\hat{h} = O(1/(M^2 + N^2))$ so, formally, the preconditioner is hardly applicable to implicit time integrators. The preconditioners of first and second order are:

$$X = \mathcal{B} + \hat{h}\beta(c_1 D_1 \mathcal{B} + \mathcal{B} c_2 D_2^T) + O(\hat{h}^2 \cdot \max(M^4, N^4))$$

$$X = \mathcal{B} + \hat{h}\beta(c_1 D_1 \mathcal{B} + c_2 \mathcal{B} D_2^T) + (\hat{h}\beta)^2 (c_1^2 D_1^2 \mathcal{B} + 2c_1 c_2 D_1 \mathcal{B} D_2^T + \mathcal{B} c_2^2 D_2^{T2})$$

$$+ O(\hat{h}^3 \cdot \max(M^6, N^6)).$$

Numerical experiments have shown that this type of preconditioner can be used for implicit integrators, but the performance is not very good because of the stepsize restriction.

We are now going to investigate another class of preconditioners, namely, the tensor product type or dimensional splitting type. The advantage of using these preconditioners is fairly obvious; only "one-dimensional problems" have to be solved.

Consider the solution of $P_2 x = b$ and we will now use a part of $I - \hat{h}\beta J_1 - \hat{h}\beta J_2$ where $J_1 = -c_1 I_N \otimes D_1$ and $J_2 = -c_2 D_2 \otimes I_M$. We can write

$$I - \hat{h}\beta J_1 - \hat{h}\beta J_2 = I - \hat{h}\beta J_2 (I + J_2^{-1} J_1) = I - \hat{h}\beta J_2 \left(I + \frac{c_1}{c_2} D_2^{-1} \otimes D_1 \right).$$

If $\frac{|c_1|}{|c_2|} \|D_2^{-1} \otimes D_1\|$ is small, i.e., $|c_2| \gg |c_1|$, we can approximate the last factor with the identity matrix and instead solve the "one-dimensional" system

$$(I - \hat{h}\beta J_2) x = b,$$

which we will consider in more detail below.

If c_1 is of the same magnitude as c_2, we may also use a splitting approach. Let

$$(I - \hat{h}\beta J_1 - \hat{h}\beta J_2)^{-1} = (I - (I - \hat{h}\beta J_2)^{-1} \hat{h}\beta J_1)^{-1} (I - \hat{h}\beta J_2)^{-1}$$

by Woodbury's formula. The splitting is done as follows.

Find $\alpha \in \mathbb{R}$ such that:

$$I - \alpha \hat{h}\beta J_1 \approx I - (I - \hat{h}\beta J_2)^{-1} \hat{h}\beta J_1. \tag{4.8}$$

Assume that $M = N$. The eigenvalues of the matrix on the right-hand side are:

$$\lambda_{ij} = 1 + \frac{\hat{h}\beta c_1 \mu_i}{1 + \hat{h}\beta c_2 \bar{\mu}_j}$$

where μ_i and $\bar{\mu}_j$ are the eigenvalues of $I_N \otimes D_1$ and $D_2 \otimes I_M$, respectively. One way of choosing α is then:

$$\alpha = \frac{1}{1 + \hat{h}\beta c_2 \rho(D_2)}. \tag{4.9}$$

The eigenvalues of $(I - \alpha\hat{h}\beta J_1) \cdot (I - \hat{h}\beta J_2)$ are:

$$1 + \hat{h}\beta c_2 \bar{\mu}_j + \hat{h}\beta \mu_i c_1 \frac{1 + \hat{h}\beta c_2 \bar{\mu}_j}{1 + \hat{h}\beta c_2 \rho(D_2)}.$$

This approximation will be good for large eigenvalues (close to $\rho(D_2)$) and not so good for small eigenvalues. However, the large eigenvalues will normally be the most troublesome, so the preconditioner should work reasonably well. If $c_1 \gg c_2$, we can solve the linear system by using the matrix $I - \hat{h}\beta J_1$ only.

Another preconditioner that approximates B is the tensor product type:

$$\tilde{P}_2 = (I + \hat{h}\beta c_1 (I_N \otimes D_1)) \cdot (I + \hat{h}\beta c_2 (D_2 \otimes I_M)) = \tilde{P}_{21} \cdot \tilde{P}_{22}.$$

This type of operator splitting preconditioner is also suggested in [3] for reaction-transport equations.

Solving linear systems $\tilde{P}_2 x = b$ is a two-stage process; first solve $\tilde{P}_{21} y = b$, then solve $\tilde{P}_{22} x = y$. Iterative methods could be used for these linear systems, but we show in the following that direct methods can be applied, and the cost is not larger than for the matrix-vector product Bv shown previously.

First consider $\tilde{P}_{21} y = b$, and assume that we have factored the one-dimensional Newton matrix:

$$I_M + \hat{h}\beta c_1 D_1 = \bar{L} \cdot \bar{U},$$

which takes $\Theta(M^3)$ operations. Note that this also applies to (4.8). Then

$$\tilde{P}_{21} = I_N \otimes (I_M + \hat{h}\beta c_1 D_1) = I_N \otimes \bar{L} \cdot \bar{U} = (I_N \otimes \bar{L}) \cdot (I_N \otimes \bar{U}).$$

Since $(I_N \otimes \bar{L})$ and $(I_N \otimes \bar{U})$ are block diagonal matrices with lower, respectively, upper triangular blocks of size M, we can solve $\tilde{P}_{21} y = b$ by doing forward and backward substitution simultaneously on all blocks of y (the blocks are of size M). This amounts to $\Theta(M^2)$ operations per block and $\Theta(M^2 N)$ operations for the complete process. Note that this process is ideal for parallel computation. We see this clearly if we consider $(I_N \otimes \bar{L}) y = b$ with $y = \text{vec}(\mathcal{Y})$ and $b = \text{vec}(\mathcal{B})$ giving the equation $\bar{L}\mathcal{Y} = \mathcal{B}$. This is a matrix equation for $\mathcal{Y}$ and we can solve it by operating on all columns in parallel.

To solve $\tilde{P}_{22} x = y$ we assume that

$$(I_N + \hat{h}\beta c_2 D_2) = L \cdot U;$$

hence,

$$\tilde{P}_{22} = (I_N + \hat{h}\beta c_2 D_2) \otimes I_M = L \cdot U \otimes I_M = (L \otimes I_M) \cdot (U \otimes I_M).$$

The matrix $(L \otimes I_M)$ is $N \times N$ block lower triangular and block (i, j) is $l_{ij} I_M$. If we partition a vector $v \in \mathbb{R}^{MN}$ into blocks of size N, we can solve $(L \otimes I_M)v = b$ by performing the forward substitution process treating each block of v as a single unknown. The matrix version of this is $\mathcal{V} L^T = \mathcal{B}$, where $v = \text{vec}(\mathcal{V})$ and $b = \text{vec}(\mathcal{B})$. The backward substitution is done in the same way. Both processes take $\Theta(MN^2)$ operations.

Note that if $M = N$, we only need to store one differentiation matrix, but we would still have to store and factor two Newton matrices since $c_1 \neq c_2$ in general. We can save space by using $(I/\hat{h}\beta c_i) - J_i$ as the Newton matrices instead of $I - \hat{h}\beta c_i J_i$.

We use the same updating strategy for the Newton matrix as was applied in the one-dimensional case, reevaluate and refactor only when the convergence of the iterative linear solver demands it. The only difference here is that we have to reevaluate and refactor two Newton matrices.

We see that

$$\tilde{P}_2 - P_2 = c_1 c_2 (\hat{h}\beta)^2 (I_N \otimes D_1) \cdot (D_2 \otimes I_M)$$

so $\|\tilde{P}_2 - P_2\| = |c_1|\,|c_2|\,O(\hat{h}^2 M^2 N^2)$ which is not small except for $\hat{h}$ very small. Hence in stiff regions where $\hat{h} = O(1)$ the splitting is not an efficient preconditioner.

If $c_2 \gg c_1$ and $\hat{h} = O(1)$ we see that $\|\tilde{P}_{22}\| \gg \|\tilde{P}_{21}\|$ and we might as well solve only $\tilde{P}_{22}x = b$. This works better than doing both solution processes for an example with $c_1 = 1$ and $c_2 = 100$. Note that the procedure for solving only for $\tilde{P}_{22}x = b$ is in agreement with the condition for solving only for $(I - \hat{h}\beta J_2)x = b$ in the method of direct inversion described above. A possible computational scheme is to use the method of direct inversion when the stepsize $\hat{h}$ is expected to be relatively large, and to use the splitting method when $\hat{h}$ is expected to remain small throughout the integration interval.

5. A hyperbolic system in two space dimensions. We now go on to study preconditioning for the linear system of PDEs:

$$u_t + A_1 u_x + A_2 u_y = 0, \quad u \in \mathbb{R}^k, \quad A_1, A_2 \in \mathbb{R}^{k\times k},$$

which, after spectral discretization, becomes:

$$V_t + A_1 \otimes (I_N \otimes D_1)V + A_2 \otimes (D_2 \otimes I_M)V = 0, \qquad V \in \mathbb{R}^{kMN}$$

where A_1 and A_2 are constant coefficient matrices.

The Newton matrix is:

$$B = I_{kNM} + \hat{h}\beta(A_1 \otimes (I_N \otimes D_1) + A_2 \otimes (D_2 \otimes I_M)),$$

and we will again consider tensor product type of preconditioners. A preconditioner based on operator splitting is:

$$\tilde{P}_3 = [I_{kMN} + \hat{h}\beta A_1 \otimes (I_N \otimes D_1)] \cdot [I_{kMN} + \hat{h}\beta A_2 \otimes (D_2 \otimes I_M)].$$

Linear systems with $\tilde{P}_3$ can be solved iteratively; however, solution of $\tilde{P}_3 x = b$ can also be done by direct methods.

Let $\tilde{P}_3 = \tilde{P}_{31} \cdot \tilde{P}_{32}$. We factor $\tilde{P}_{32}$ as follows:

$$\begin{aligned}\tilde{P}_{32} &= I_{kMN} + \hat{h}\beta A_2 \otimes D_2 \otimes I_M \\ &= (I_{kN} + \hat{h}\beta A_2 \otimes D_2) \otimes I_M \\ &= L \cdot U \otimes I_M = (L \otimes I_M) \cdot (U \otimes I_M).\end{aligned}$$

Solution of linear systems with $\tilde{P}_{32}$ can be performed in the same way as described for $\tilde{P}_{22}$ above. A more efficient approach in the case where A_2 is diagonalizable, $A_2 = T\Lambda T^{-1}$, $\Lambda = \operatorname{diag}(\lambda_i)$ is to transform $\tilde{P}_{32}x = b$ into $(I_{kMN} + \hat{h}\beta\Lambda \otimes D_2 \otimes I_M)\tilde{x} = \tilde{b}$. This linear system can be solved by factoring the k matrices $I_N + \hat{h}\beta\lambda_i D_2$ and using these LU factors in the solution process. The cost of factoring A_2 is usually negligible because the number of PDEs is small compared to M and N. Each transform by $T^{-1} \otimes I_N \otimes I_M$ costs $\Theta(k^2MN)$, and this is also negligible compared to the total cost which is $\Theta(kN^3 +$

kMN^2). The major contributions are the factoring of the k matrices $I + \hat{h}\beta\lambda_i D_2$ of size N and the solves for the total linear system, each costing $\Theta(kMN^2)$ operations.

If A_1 is diagonalizable by T_1: $A_1 = T_1\Lambda_1 T_1^{-1}$, $\Lambda_1 =$ diag(λ_{1i}) we can write $\tilde{P}_{31}x = b$ as

$$(I_{kMN} + \hat{h}\beta\Lambda_1 \otimes I_N \otimes D_1)\tilde{x} = \tilde{b}$$

where $\tilde{x} = (T_1^{-1} \otimes I_{MN})x$, $\tilde{b} = (T_1^{-1} \otimes I_{MN})b$. The transformed linear system decouples into k blocks, each having the same form as $P_{21}x = b$ considered in the previous section, and can be solved by factoring the k matrices $I_M + \hat{h}\beta\lambda_{1i}D_1$ and using a blockwise substitution process as described for the solution of $P_{21}x = b$. The total cost of factoring and solving is $\Theta(kM^3 + kM^2N)$ operations. If k, the number of PDEs, is large, we can use a more efficient procedure. If

$$I_M + \hat{h}\beta\lambda_{11}D_1 = S\,H\,S^{-1},$$

then

$$I_M + \hat{h}\beta\lambda_{1i}D_1 = \frac{\lambda_{1i}}{\lambda_{11}} S\left(\left(\frac{\lambda_{11}}{\lambda_{1i}} - 1\right)I + H\right)S^{-1}.$$

If S is chosen such that H is diagonal, then no additional factoring is necessary for $I_M + \hat{h}\beta\lambda_{1i}D_1, i = 2, \ldots, k$. If H is upper Hessenberg, then factoring of $((\lambda_{11}/\lambda_{1i})-1)I+H$ is required, but this is cheaper ($\Theta(kM^2)$ operations compared to $\Theta(kM^3)$) than factoring the matrices $I_M + \hat{h}\beta\lambda_{1i}D_1$ for $i = 2, \ldots, k$.

We may also use the direct inversion scheme for linear two-dimensional systems. Let

$$P_3 = I - \hat{h}\beta J_1 - \hat{h}\beta J_2$$

where $J_1 = -A_1 \otimes I_N \otimes D_1$ and $J_2 = -A_2 \otimes D_2 \otimes I_M$.

As for the single two-dimensional equation, we solve only for $(I - \hat{h}\beta J_2)x = b$ if $\|J_2\| \gg \|J_1\|$, i.e., $\rho(A_2) \gg \rho(A_1)$ (if we assume $M = N$). Even if $\rho(A_2) \approx \rho(A_1)$, numerical experiments show that the one-dimensional preconditioner will work reasonably well at least in the cases where $M = N$. When $\rho(A_2) \approx \rho(A_1)$ we may use the approximation

$$P_3 = (I - \hat{h}\beta\alpha J_1)(I - \hat{h}\beta J_2)$$

with

$$\alpha = \frac{1}{1 + \hat{h}\beta\rho(A_2 \otimes D_2 \otimes I_M)} = \frac{1}{1 + \hat{h}\beta\rho(A_2)\rho(D_2)}.$$

The performance of the preconditioner will depend on the spectra of A_1 and A_2, and may therefore be less predictable than for the single two-dimensional equation.

6. Remarks on quasi-linear PDEs. For nonlinear systems of PDEs the Jacobian will no longer be constant, and an approximation to the Jacobian has to be computed several times during the integration to make the ODE solver work properly. The corresponding Newton matrix can be operated on without forming it explicitly, similar to the approach for linear systems shown previously.

Quasi-linear systems comprise an important special case, and for these equations we can do a local linearization and recompute the preconditioner when the dependent variables and/or the stepsize have changed significantly. The control mechanism for recomputing the Jacobian is still the convergence of the linear solver. With the local linearization we can solve the ODE system in the same way as described previously for linear systems of PDEs.

As an illustration, consider the spectral discretization of the PDE

$$u_t + u\,u_x = 0$$

which can be written:

$$V_t + V \odot D \cdot V = 0, \qquad V \in \mathbb{R}^N,$$

or

$$V_t + \operatorname{diag}(V) \cdot D \cdot V = 0 \tag{6.1}$$

where $\odot$ denotes the componentwise product of two vectors, and $\operatorname{diag}(V)$ is the diagonal matrix formed by the components of the vector V. The Jacobian is formed by taking the Fréchet derivative of $\operatorname{diag}(V) \cdot D \cdot V$. The result is:

$$J = -\operatorname{diag}(V) \cdot D - \operatorname{diag}(D \cdot V).$$

We see that J depends directly on the approximate solution vector V. The spectral discretization of a local linearization of the PDE gives:

$$V_t + \operatorname{diag}(\hat{V}) \cdot D \cdot V = 0$$

where $\hat{V}$ is the point of linearization. The Jacobian in this case is obviously $-\operatorname{diag}(\hat{V}) \cdot D$, which may not be a good approximation to the exact Jacobian. A better choice would be to insert $\hat{V}$ into the formula for the exact Jacobian.

Now consider the quasi-linear system

$$u_t + A(u)\,u_x = 0, \quad u \in \mathbb{R}^k, \quad A(u) = \{a_{ij}(u)\}. \tag{6.2}$$

If $A(u) = A$ is a constant matrix, this problem reduces to the one-dimensional model problem considered in §3. Spectral discretization with N points gives the following ODE system:

$$V_t + \{\operatorname{diag}(a_{ij}(V))\}_{i,j=1}^{k} \cdot (I_k \otimes D) \cdot V = 0, \qquad V \in \mathbb{R}^{kN} \tag{6.3}$$

where $a_{ij}(V)$ is interpreted as follows: In the continuous case a_{ij} is a function of all k components of u, and the value of a_{ij} is a real number. In the discrete case a_{ij} takes the k N-vectors of V as arguments and produces an N-vector. The expression $\{\operatorname{diag}(a_{ij}(V))\}_{i,j=1}^{k}$ is therefore a matrix consisting of diagonal matrices of the N-vectors a_{ij}. We see that this spectral discretization formula applied to the single PDE above gives (6.1). In order to compute the Jacobian for the ODE system we take the Fréchet derivative of

$$\{\operatorname{diag}(a_{ij}(V))\}_{i=1}^{k} \cdot (I_k \otimes D) \cdot V.$$

This gives after some algebraic manipulations:

$$J = -\{\operatorname{diag}(a_{ij}(V))\}_{i,j=1}^{k} \cdot (I_k \otimes D) - \left\{\operatorname{diag}\left(\sum_{m=1}^{k} DV_m \odot \frac{\partial a_{im}}{\partial V_j}\right)\right\}_{i,j=1}^{k} \tag{6.4}$$

where $V = [V_1, \ldots, V_k]^T$, $V_m \in \mathbb{R}^N$, and $\partial a_{ij}/\partial V_j$ have values in $\mathbb{R}^N$. The Jacobian is a $Nk \times Nk$ matrix. Preconditioners for J can be constructed as shown in the previous sections.

Example. Nonlinear wave propagation in the atmosphere can be modelled by the following quasi-linear PDE system:

$$u_t + A(u)\, u_x = b$$

where

$$A(u) = \begin{bmatrix} u_2 & u_1 & 0 \\ 0 & u_2 & 1/u_1 \\ 0 & \gamma u_3 & u_2 \end{bmatrix}, \qquad \gamma \in \mathbb{R}.$$

Spectral discretization gives:

$$V_t + \begin{bmatrix} \operatorname{diag}(V_2) & \operatorname{diag}(V_1) & 0 \\ 0 & \operatorname{diag}(V_2) & \operatorname{diag}(1/V_1) \\ 0 & \gamma \operatorname{diag}(V_3) & \operatorname{diag}(V_2) \end{bmatrix} \cdot (I_3 \otimes D) \cdot V = B. \tag{6.5}$$

Using the previous formula for the Jacobian, we get the following result:

$$\begin{aligned} J = &- \begin{bmatrix} \operatorname{diag}(V_2) & \operatorname{diag}(V_1) & 0 \\ 0 & \operatorname{diag}(V_2) & \operatorname{diag}(1/V_1) \\ 0 & \gamma \operatorname{diag}(V_3) & \operatorname{diag}(V_2) \end{bmatrix} \cdot (I_3 \otimes D) \\ &- \begin{bmatrix} \operatorname{diag}(DV_2) & \operatorname{diag}(DV_1) & 0 \\ -\operatorname{diag}(DV_3 \odot 1/V_1^2) & \operatorname{diag}(DV_2) & 0 \\ 0 & \operatorname{diag}(DV_3) & \gamma \operatorname{diag}(DV_2) \end{bmatrix}. \end{aligned} \tag{6.6}$$

For the two-dimensional quasi-linear system

$$u_t + A_1(u)\, u_x + A_2(u)\, u_y = b, \qquad u \in \mathbb{R}^k,$$

we get, after spectral discretization:

$$\begin{aligned} V_t &+ \left\{\operatorname{diag}(a_{ij}^{(1)}(V))\right\} \cdot (I_k \otimes I_N \otimes D_1) V \\ &+ \left\{\operatorname{diag}(a_{ij}^{(2)}(V))\right\} \cdot (I_k \otimes D_2 \otimes I_M) V = B, \qquad V \in \mathbb{R}^{MNk} \end{aligned}$$

because the differentiation matrices are $I_N \otimes D_1$ and $D_2 \otimes I_M$ in the x- and y-direction, respectively. Using the same procedure as in the one-dimensional case, we get the following Jacobian:

$$\begin{aligned} J = &- \left\{\operatorname{diag}(a_{ij}^{(1)}(V))\right\}_{i,j=1}^{k} \cdot (I_k \otimes I_N \otimes D_1) \\ &- \left\{\operatorname{diag}\left(\textstyle\sum_{m=1}^{k} (I_N \otimes D_1) V_m \odot \frac{\partial a_{im}^{(1)}}{\partial V_j}\right)\right\}_{i,j=1}^{k} \\ &- \left\{\operatorname{diag}(a_{ij}^{(2)}(V))\right\}_{i,j=1}^{k} \cdot (I_k \otimes D_2 \otimes I_M) \\ &- \left\{\operatorname{diag}\left(\textstyle\sum_{m=1}^{k} (D_2 \otimes I_M) V_m \odot \frac{\partial a_{im}^{(2)}}{\partial V_j}\right)\right\}_{i,j=1}^{k} \end{aligned}$$

where $V = [V_1, \ldots, V_k]^T$, $V \in \mathbb{R}^{MN}$ and $\partial a_{im}^{()}/\partial V_j$ have values in $\mathbb{R}^{MN}$. Preconditioners for J can be constructed in the same way as before. The Jacobian is relatively complex, and it may be possible to use a preconditioner that uses only the most significant terms from the Jacobian. However, this simplification will depend entirely on the actual system to be solved.

Refactoring of the Newton matrix $I - \hat{h}\beta J$ requires more monitoring for a nonlinear system since we have to decide if the Jacobian should be reevaluated or if we should just use the current stepsize with the old Jacobian. Practice among existing codes varies considerably. The LSODE family always reevaluates the Jacobian when convergence failure occurs, whereas SIMPLE reevaluates only when the iteration using the current stepsize in the Newton matrix diverges.

Stiffness is normally not a problem in nonlinear equations like the simple model equation above, but quasi-linear systems coming from, e.g., gas dynamics and wave propagation, show considerable stiffness. Explicit ODE solvers work very inefficiently for such systems, and implicit solvers could be more efficient as we have seen for the linear test problems.

7. Numerical examples. In this section we will give a few examples to show that the preconditioners developed in the previous sections work reasonably well. More extensive testing is, however, required in order to assess the true quality of the preconditioners. All the tests reported have been run on VAX stations using double precision arithmetic.

Consider the two-dimensional equation treated in §4:

$$u_t + c_1 u_x + c_2 u_y = 0, \qquad c_1 > 0, c_2 > 0$$

in $[-1, 1] \times [-1, 1]$ with the following boundary conditions: $u(-1, y, t) = 0, u(x, -1, t) = 0$, and open boundaries along $x = 1$ and $y = 1$. The initial function is the "Gaussian hill" $\exp(-10(x^2 + y^2))$ and the integration interval is $[0, 2]$. Both the absolute and the relative error tolerance were set to 10^{-4}.

In Table 7.1 we give some results for this equation using the ODE solver KRYSI, varying the number of spectral points and the "wave speeds" c_1 and c_2. The preconditioner used is the "one-dimensional" version: $I - \hat{h}\beta J_2$ based on the preconditioning matrix Q.

TABLE 7.1
ODE *statistics for* KRYSI *applied to a two-dimensional model equation.*

M=N	9	9	9	17	17	17
c_1	1	1	1	1	1	1
c_2	10	100	1000	10	100	1000
steps	132	130	137	102	100	105
fixp. steps	116	109	109	73	73	73
func. eval.	1285	1285	1383	975	961	1043

We see that the number of steps and the number of function evaluations is not increasing with N (it is even decreasing), and that the number of fixpoint steps is large compared to the total number of steps. This indicates that the ODE system is not very stiff, but what is happening is the following: As long as the wave is inside the region (corresponds to $t \leq 1$) the stepsize is relatively small due to the tracking of the relatively steep wave. Mainly fixpoint steps are used in this time interval. When the wave has left the region, the solver switches to Newton steps, and the stepsize increases sharply.

This is illustrated in Table 7.2 where we have used the case $M = N = 9, c_1 = 1, c_2 = 100$ and variable integration interval. The performance of KRYSI on this model problem is good, the internal statistics of the solver show that there are few, if any, convergence failures in the linear solver, and 2–3 nonlinear iterations are used for each step. The cost of preconditioning for the cases shown in Table 7.1 is low. There are 4–6 evaluations and factorizations of the preconditioner for each case.

TABLE 7.2
Number of steps as a function of integration interval.

t_{end}	2.0	10.0	100.0
steps	130	137	143
fixp. steps	109	109	109

As a comparison to KRYSI we have used the explicit solver DOPRI5 on some of the cases displayed in Table 7.1. We see clearly from Table 7.3 that the explicit solver uses many more steps and function evaluations than KRYSI. The right-most case in the table was aborted after $\approx$ 25000 steps.

TABLE 7.3
Statistics for DOPRI5 *on the two-dimensional model problem.*

$M = N$	9	9	17
c_1	1	1	1
c_2	10	100	1
steps	344	2094	$\approx \infty$
func. eval.	2408	14658	$\approx \infty$

When using the modified preconditioner

$$(I - \alpha\hat{h}\beta J_1)(I - \hat{h}\beta J_2)$$

with $c_1 = c_2$, we got results that were almost identical to those obtained with the "one-dimensional" preconditioner. This can be partly explained by looking at the eigenvalues λ_{ij} of the matrix $(I - \hat{h}\beta J_1)^{-1}(I - \hat{h}\beta J_1 - \hat{h}\beta J_2)$ in the $M = N$ case:

$$\lambda_{ij} = (1 + \hat{h}\beta c_2\mu_i)^{-1}(1 + \hat{h}\beta c_1\mu_i + \hat{h}\beta c_2\bar{\mu}_j).$$

If we take $c_1 = c_2$, we get $\lambda_{ij} = 1 + a_j/(1 + a_i)$ where $a_i = \hat{h}\beta c_1\mu_i$, and we see that the location of the spectrum depends on a_i. If $a_i = O(1)$, i.e., $\hat{h}\beta c_1 = O(N^{-2})$, then the spectrum will be fairly close to $(1, 0)$. A preliminary recommendation will therefore be to use the "one-dimensional" preconditioner $I - \hat{h}\beta J_2$ for $c_2 \geq c_1$ and to use $(I - \hat{h}\beta J_1)$ for $c_1 > c_2$.

We have also run the same model equation on the ODE solver LSODPK with the "one-dimensional" preconditioner and using the same settings for the ODE parameters. Some results are given in Table 7.4:

TABLE 7.4
Statistics for LSODPK *applied to the model problem.*

$M = N$	9	9	9	17	17
c_1	1	1	1	1	1
c_2	1	10	100	1	10
steps	67	187	1363	73	377
func. eval.	202	608	4329	201	1400

We see that the performance is not satisfactory, especially when c_2 is large. It is well known that the BDF family of methods, on which LSODPK is based, is not A-stable for order > 2; see [10] for plots of the stability regions. The spectrum of the differentiation matrix is discussed in [25], and there are eigenvalues close to the imaginary axis, which can cause stability problems, and hence the solver will use an unnecessary small stepsize. A further complication is the high sensitivity of the eigenvalues of the differentiation matrix to machine precision (see [25]), and it is reasonable to expect that the stability restrictions will be a function of the machine precision.

When inspecting the order used by the ODE solver in the integration of the examples in Table 7.4, it was discovered that order four was used in most of the steps, and even order five was used sometimes. So by looking at the stability regions for these methods it is fairly clear that stability did play a role in the performance of LSODPK shown in Table 7.4.

Fortunately, LSODPK contains an option to put an upper bound on the order to be used by the solver. We set this maximum value to three, so the resulting family of methods would be almost A-stable. This gave much better results as shown in Table 7.5.

TABLE 7.5
Statistics for LSODPK *with maximum order* = 3.

$M = N$	9	9	9	17	17	17
c_1	1	1	1	1	1	1
c_2	10	100	1000	10	100	1000
steps	133	122	120	98	94	71
func. eval.	429	382	363	306	299	219
ave. N.	1.14	1.15	1.13	1.16	1.12	1.14
ave. lin.	1.83	1.72	1.66	1.70	1.83	1.69
pre. eval.	16	17	17	14	14	14
CPU time	3.5	3.2	3.2	4.1	5.0	5.1

We see that, compared to KRYSI, the number of steps is approximately equal for the test cases, but the number of function evaluations is much smaller. In Table 7.5 we have included more statistics in order to show the performance of the preconditioner. The parameter "ave. N." is the average number of Newton iterations per step, and this is indeed small for these tests. The parameter "ave. lin." is the average number of linear iterations per Newton step and this is probably the best measure on the performance of the preconditioner. The numbers are very small indicating that the preconditioner works well. Another indication is that there were no convergence failures of the linear iteration in any of the tests. The parameter "pre. eval." is the number of preconditioner evaluations which, in this case, is also the number of Jacobian evaluations. Analytical Jacobians were used for both ODE solvers, so Jacobian evaluations do not show up in the function evaluation figure.

Restricting the order to 2 in LSODPK did not produce better results than shown for maximum order = 3, nor did runs with maximum order = 4.

The next test example is the one-dimensional atmosphere model whose spectral discretization is given in (6.5). The Jacobian matrix for the ODE system is given in (6.6). Since we know the exact Jacobian, we can, in principle, compute the exact Newton matrix for preconditioning. But computing the Newton matrix for each change in stepsize is normally too costly, and the LSODPK solver which is used in this example tries to minimize the number of Newton matrix computations. Convergence of the Newton iteration may be bad for "old" Jacobians and for increasing stepsizes as the analysis in §3 indicates.

The Newton matrix in this case is $I - \bar{h}\beta J$ where J is the Jacobian in (6.6), and we use the matrix Q in the construction of the preconditioner. We see that the Jacobian contains two terms, the first term is what is obtained if local linearization is done before differentiation, while the second term is a true property of quasi-linear systems of first order PDEs.

Two versions of the preconditioner have been investigated in the tests. The first version is using the full Jacobian, while the second version is using only the first term. It turns out that the latter version gives the best results, and is therefore used in this test example. The difference between the two versions of the preconditioner may be due to the fact that Jacobians are computed only when needed to assure convergence of the iterations. Meanwhile the variables may have changed so much that the second term, which depends strongly on the values of the variables, is making the convergence worse than if the term was not there.

Note that the structure of the Jacobian as well as the Newton matrix is as follows (cf. (6.6)):

$$\begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{11} & 0 \\ 0 & A_{32} & A_{33} \end{bmatrix}.$$

In order to solve $(I - \hat{h}\beta J)x = b$, we have used the capacitance matrix technique in [15].

Boundary treatment in spectral methods for hyperbolic systems has been discussed in [5] for fixed stepsize and Dirichlet boundary conditions. In our case we have open boundary conditions and variable stepsize.

To address the issue of boundary conditions properly, we perform a diagonalization of the PDE system. We get the following eigenvalues (=characteristic speeds): $u_2 - a, u_2, u_2 + a$ where a is the local speed of sound: $a = \sqrt{\gamma u_3/u_1}$. The corresponding characteristic variables are $u_3 - u_1 a u_2, u_1 - u_3/a^2, u_3 - u_1 a u_2$. Assuming subsonic conditions, we get one left-going characteristic, one right-going, and one characteristic whose direction depends on $\text{sign}(u_2)$. Following the principles for boundary treatment in [5], we compute the first characteristic variable at the left boundary, the third characteristic variable at the right boundary, and the second characteristic variable at the boundary points determined by $\text{sign}(u_2)$. For inflow boundary points we supply appropriate inflow values. From both outflow and inflow characteristic variables we compute the boundary values for the physical variables.

The initial conditions are given by hydrostatic equilibrium and a state equation:

$$u_{3,x} = -u_1\, g, \qquad u_3\, u_1^{\gamma-1} = \text{constant},$$

plus a small perturbation in u_3 of the form $0.1 \cdot \exp(-50.0 * x^2)$.

The test results for an explicit ODE solver (DOPRI5) and the implicit solver LSODPK, both with error tolerances set to 10^{-4}, are shown in the Tables 7.6 and 7.7. We see that the number of steps is $\Theta(N^2)$ for DOPRI5, while for LSODPK the number of steps is only slowly varying with N. The performance of the preconditioner is still satisfactory, but not so good as for the single two-dimensional hyperbolic PDE shown in Table 7.5. The difference in CPU time between DOPRI5 and LSODPK is large for large N and, in particular, for longer integration intervals even if the stiffness is relatively moderate here. However, we might have used a cheaper explicit method since DOPRI5 has seven stages and the stepsize is determined by the stability requirement. For example, a low order Runge–Kutta method or an Adams method would probably do better in this test example. The number of function evaluations for LSODPK is reasonably low, and using this solver with a good preconditioner on the atmosphere model seems to be an efficient method. Hopefully, the method can be used to solve other important quasi-linear hyperbolic systems with the same efficiency.

TABLE 7.6
Statistics for LSODPK *on the atmosphere model.*

t_{end}	1.0	1.0	1.0	4.0	4.0	4.0
N	17	33	65	17	33	65
steps	57	74	74	114	154	118
func. eval.	231	385	427	531	874	739
ave. N.	1.64	1.60	1.62	1.53	1.51	1.57
ave. lin.	2.05	1.97	2.09	1.85	1.81	1.79
npe	10	11	10	12	12	12
CPU time	6.2	10.3	11.5	14.3	23.5	19.8

TABLE 7.7
Statistics for DOPRI5 *on the atmosphere model.*

t_{end}	1.0	1.0	1.0	4.0	4.0	4.0
N	17	33	65	17	33	65
steps	59	217	873	184	711	2837
func. eval.	413	1519	6111	1288	4977	19860
CPU time	5.5	20.2	81.4	17.1	66.3	264.5

8. Conclusion.

We have shown how to apply implicit ODE solvers with iterative linear equation solvers on ODEs coming from spectral discretization of some hyperbolic model problems. The tensor product-type developed preconditioners perform satisfactorily, and the use of implicit ODE solvers gives a significant reduction in the number of timesteps and computing time compared to explicit ODE solvers. In cases where the integration interval is long, the advantage of using implicit solvers seems to be substantial.

The application to quasi-linear PDEs is straightforward (by local linearization), but because of the nonlinearity the preconditioners may be less efficient if the solution is rapidly changing. A test example from atmospheric physics shows that application of implicit ODE solvers to quasi-linear PDE systems gives promising results. We will describe preconditioning techniques for quasi-linear systems of PDEs in more detail in a forthcoming paper.

For PDEs containing second order derivatives, different types of preconditioners have been suggested. Orszag [20] has derived a preconditioner based on centered finite differences of the Chebyshev points $\{x_j\}$, and that preconditioner has proven to be very effective, see, e.g., [12]. The numerical experiments indicates that the spectral radius of the preconditioned second derivative matrix is $\Theta(1)$ compared to $\Theta(N^4)$ for the unpreconditioned case. However, the use of these preconditioners in the ODE context will meet the same problems as for the preconditioners described in this paper.

Note that preconditioning of the second derivative matrix is in many ways simpler than for the first derivative matrix due to the fact that the latter matrix is nearly defective. The paper [26] discusses in detail the eigenvalues of the second derivative matrix.

REFERENCES

[1] R. BELLMAN, *Introduction to Matrix Analysis*, Tata-McGraw-Hill, New Delhi, 1974.

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

[3] ———, *Reduced-storage matrix methods in stiff ODE systems*, J. Comput. Appl. Math., 31 (1989), pp. 40–91.

[4] P. N. BROWN AND Y. SAAD, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 450–481.

[5] C. CANUTO AND A. QUARTERONI, *On the boundary treatment in spectral methods for hyperbolic systems*, J. Comput. Phys., 71 (1987), pp. 100–110.

[6] T. F. CHAN AND K. R. JACKSON, *The use of iterative linear equation solvers in codes for large systems of stiff* IVPs *for* ODEs, SIAM J. Sci. Statist. Comp., 7 (1986), pp. 378–417.

[7] M. DEVILLE, L. KLEISER, AND F. MONTIGNY-RANNOU, *Pressure and time treatment for Chebyshev spectral solution of a Stokes problem*, Int. J. Numer. Meth. Fluids, 4 (1984), pp. 1149–1163.

[8] D. FUNARO, *A preconditioning matrix for the Chebyshev differencing operator*, SIAM J. Numer. Anal., 24 (1987), pp. 1024–1031.

[9] ———, *Computing the inverse of the Chebyshev collocation derivative*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 1050–1057.

[10] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[11] C. W. GEAR AND Y. SAAD, *Iterative solution of linear equations in* ODE *codes*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 583–601.

[12] P. HALDENWANG, G. LABROSSE, S. ABBOUDI, AND M. DEVILLE, *Chebyshev* 3D *spectral and* 2D *pseudospectral solvers for the Helmholtz equation*, J. Comput. Phys., 55 (1984), pp. 115–128.

[13] E. HAIRER, S. P. NØRSETT, AND G. WANNER, *Solving Ordinary Differential Equations* I, Springer-Verlag, Berlin, 1987.

[14] A. C. HINDMARSH AND S. P. NØRSETT, KRYSI, *an* ODE *solver combining a semi-implicit Runge–Kutta method and a preconditioned Krylov method*, Mathematics and Computation report 8/87, Department of Numerical Math., University of Trondheim, Norway, 1987.

[15] A. KARAGEORGHIS AND T. N. PHILLIPS, *Chebyshev spectral collocation methods for laminar flow through a channel contraction*, J. Comput. Phys., 84 (1989), pp. 114–133.

[16] P. LANCASTER AND M. TISMENETSKY, *Theory of Matrices*, Academic Press, New York, 1985.

[17] A. LERAT, *Une classe de schémas implicites pour les systèmes hyperboliques de lois de conservation*, C. R. Acad. Sci. Paris Ser. A, (1979), pp. 1033–1036.

[18] R. E. LYNCH, J. R. RICE, AND D. H. THOMAS, *Direct solution of partial difference equations by tensor product methods*, Numer. Math., 6 (1964), pp. 185–199.

[19] J. OUAZZINI, R. PEYRET, AND A. ZAKARIA, *Stability of collocation-Chebyshev schemes with application to Navier–Stokes equations*, Proc. 6th GAMM Conf. Numer. Meth. Fluid Mech., D. Rues and W. Kordulla, eds., Vieweg, Braunschweig, 1985, pp. 287–294.

[20] S. ORSZAG, *Spectral methods for problems in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70–92.

[21] S. C. REDDY AND L. N. TREFETHEN, *Lax-stability of fully discrete spectral methods via stability regions and pseudo-eigenvalues*, Proc. ICOSAHOM Conf., 1989, C. Canuto and A. Quarteroni, eds., North-Holland, Amsterdam, 1990, pp. 147–164.

[22] P. E. SAYLOR AND R. SKEEL, *Linear iterative methods for implicit* ODE *methods*, ICASE Report No. 90-51, NASA Langley Research Center, VA, 1990.

[23] T. D. TAYLOR, R. S. HIRSH, AND M. M. NADWORNY, *Comparison of* FFT, *direct inversion and conjugate gradient methods for use in pseudospectral methods*, Comput. & Fluids, 12 (1984), pp. 1–9.

[24] P. G. THOMSEN, SIMPLE *user's guide*, Report, Dept. of Numerical Math., Danish Institute of Technology, Lyngby, Denmark, 1987.

[25] L. N. TREFETHEN AND M. R. TRUMMER, *A stability phenomenon in spectral methods*, SIAM J. Numer. Anal., 24 (1987), pp. 1008–1023.

[26] J. A. C. WEIDEMANN AND L. N. TREFETHEN, *The eigenvalues of second-order spectral differentiation matrices*, SIAM J. Numer. Anal., 25 (1988), pp. 1279–1298.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1214–1236, September 1993

METHODS FOR REDUCING APPROXIMATE-FACTORIZATION ERRORS IN TWO- AND THREE-FACTORED SCHEMES*

ERLENDUR STEINTHORSSON† AND TOM I-P. SHIH†

Abstract. Three methods are presented for reducing approximate-factorization (AF) errors that exist in two- and three-factored schemes, such as the popular ADI, LU, and LU-SSOR methods. For problems in which AF errors are larger than both time-discretization and time-linearization errors, these methods can be used to lower the AF errors so that larger time-step sizes can be used when transient solutions are of interest. When only steady-state solutions are of interest, these methods can be used to accelerate the convergence rate. The methods presented also can be used to stabilize schemes that are unstable because of AF errors (e.g., the three-factored ADI scheme applied to the linear advection equation with central-difference approximation of the spatial derivatives). The three methods presented can be added easily to existing codes using any two- or three-factored schemes.

Key words. approximate factorization, approximate-factorization errors, ADI, LU, LU-SSOR, computational fluid dynamics

AMS subject classifications. 65C20, 65MO5, 76N15

1. Introduction.

All implicit finite-difference (FD) and finite-volume (FV) methods for partial differential equations (PDEs), such as the Euler and Navier–Stokes equations, require simultaneous algebraic equations to be solved. For computational efficiency it is desirable that such methods solve only those simultaneous equations that can be analyzed efficiently, namely, those that are linear, are of as low an order as possible, and have coefficient matrices that possess certain structures, such as tridiagonal or triangular (e.g., bidiagonal) matrices. Systems of nonlinear algebraic equations can always be converted to systems of linear equations by the Newton–Raphson iteration or by variations of it. To reduce the order of simultaneous equations and to change the structure of their coefficient matrices, a technique known as approximate factorization (AF) can be used.

AF has been used in three ways. The first approach, factorization according to spatial dimensions, is used in ADI methods [1], [2] and their variations [3], [4]; schemes based on ADI factoring are two-factored for two-dimensional (2-D) problems and are three-factored for three-dimensional (3-D) problems. The second approach, factorization according to the signs of certain eigenvalues, is used in the LU [5]–[7] and LU-SSOR [8] schemes; schemes based on eigenvalue splitting are always two-factored whether the problem being analyzed is 2-D or 3-D. The third approach, which involves a combination of the first two approaches, is used in the partially split method of Steger and of Ying [9]. Note that AF should always be used with delta formulation in order to produce consistently split FD and FV methods [10].

Although AF is a useful technique for developing efficient schemes, FD and FV methods constructed by using it can have problems. For example, with ADI schemes the three-factored versions are less stable numerically than are the two-factored versions. In fact, Fourier analysis has shown that for the linear advection equation with central-difference approximation of spatial derivatives, the three-factored scheme is unconditionally unstable. With LU schemes the problem is not stability but rather the physical meaning of the intermediate variables. This makes implicit implementation of

*Received by the editors November 19, 1990; accepted for publication (in revised form) August 11, 1992. This work was supported by National Aeronautics and Space Administration grant NAG 3-929.

†Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213-3890.

boundary conditions at the intermediate step difficult. Implicit boundary conditions can significantly affect the accuracy of transient solutions and the convergence rate to steady state. Also, even though all schemes constructed by using AF wit delta formulation yield steady-state solutions that are independent of errors arising from AF, transient solutions are affected by those errors. Thus if transient solutions are of interest, then time-step sizes may need to be kept excessively small in order to minimize those errors.

If errors introduced by AF can be eliminated in an economic manner, then schemes constructed by using AF will improve in performance and accuracy. Thus the objective of this study is to devise methods that can be used to reduce or eliminate errors that arise from AF. In this paper three such methods are devised. These methods can be applied to reduce or eliminate AF errors in any scheme constructed by AF for any PDE, including the Euler and the Navier–Stokes equations. Here these three AF-error-reduction methods are presented in the framework of a three-factored ADI scheme applied to the linear advection equation.

The rest of this paper is organized as follows. First, the model equation and the algorithm used to present the AF-error-reduction methods are given. Second, three AF-error-reduction methods are described, and then one of the three AF-error-reduction methods is analyzed for its convergence and stability properties. Finally, results that show the capabilities of the three AF-error-reduction methods are given.

2. Model equation and algorithm. In this study the linear advection equation is used as the model PDE and the three-factored ADI scheme is used as the model algorithm whose AF errors are to be reduced or eliminated.

The 3-D linear advection equation is given by

$$\frac{\partial u}{\partial t} + c_x \frac{\partial u}{\partial x} + c_y \frac{\partial u}{\partial y} + c_z \frac{\partial u}{\partial z} = 0, \tag{1}$$

where $u = u(t, x, y, z)$ and c_x, c_y, and c_z are real constants. Suppose that (1) is supplied with appropriate initial and boundary conditions and that the domain is discretized so that the time-step size (Δt) and grid spacings ($\Delta x, \Delta y$, and Δz) are all constants. If the time derivative is replaced by a generalized time-differencing formula [1] and if all spatial derivatives are replaced by difference operators, (1) becomes

$$\left[1 + \frac{\gamma \Delta t}{1+\theta}(c_x D_x + c_y D_y + c_z D_z)\right] \Delta u_{ijk}^{n+1} = \mathrm{RHS}_{ijk}, \tag{2}$$

where

$$\mathrm{RHS}_{ijk} = -\frac{\Delta t}{1+\theta}(c_x D_x + c_y D_y + c_z D_z) u_{ijk}^n + \frac{\theta}{1+\theta}\Delta u_{ijk}^n. \tag{3}$$

In the above equations $\Delta u^{n+1} = u^{n+1} - u^n$ and $u_{ijk}^n = u(n\Delta t, i\Delta x, j\Delta y, k\Delta z)$, where n, i, j, and k are nonnegative integers; D_x, D_y, and D_z are the difference operators (central or upwind) that approximate the first-order x, y, and z derivatives, respectively; and θ and γ are constants that determine the time-differencing formula (e.g., $\gamma = 1$ and $\theta = 0$ give the Euler implicit formula, and $\gamma = 1$ and $\theta = \frac{1}{2}$ give the three-point backward formula).

Equation (2) can be approximately factored in several different ways. Approximate factorization according to spatial dimensions gives the following three-factored ADI scheme:

$$\left(1 + \frac{\gamma \Delta t}{1+\theta} c_x D_x\right)\left(1 + \frac{\gamma \Delta t}{1+\theta} c_y D_y\right)\left(1 + \frac{\gamma \Delta t}{1+\theta} c_z D_z\right) \Delta u_{ijk}^{n+1} = \mathrm{RHS}_{ijk}, \tag{4}$$

which can be split as

(5a) $$\left(1 + \frac{\gamma \Delta t}{1+\theta} c_x D_x\right) \Delta u^*_{ijk} = \mathrm{RHS}_{ijk},$$

(5b) $$\left(1 + \frac{\gamma \Delta t}{1+\theta} c_y D_y\right) \Delta u^{**}_{ijk} = \Delta u^*_{ijk},$$

(5c) $$\left(1 + \frac{\gamma \Delta t}{1+\theta} c_z D_z\right) \Delta u^{n+1}_{ijk} = \Delta u^{**}_{ijk}.$$

When (2) was approximately factored to produce (4), an error was introduced into (4). If (2) had been factored without introducing this error, then the following equation would have been obtained:

(6a)
$$\left(1 + \frac{\gamma \Delta t}{1+\theta} c_x D_x\right)\left(1 + \frac{\gamma \Delta t}{1+\theta} c_y D_y\right)\left(1 + \frac{\gamma \Delta t}{1+\theta} c_z D_z\right) \Delta u^{n+1}_{ijk} = \mathrm{RHS}_{ijk} + \Phi_{\mathrm{AF}},$$

where

(6b) $$\begin{aligned}\Phi_{\mathrm{AF}} = &\left(\frac{\gamma \Delta t}{1+\theta}\right)^2 [(c_x D_x)(c_y D_y) + (c_x D_x)(c_z D_z) + (c_y D_y)(c_z D_z)]\Delta u^{n+1}_{ijk} \\ &+ \left(\frac{\gamma \Delta t}{1+\theta}\right)^3 (c_x D_x)(c_y D_y)(c_z D_z)\Delta u^{n+1}_{ijk}.\end{aligned}$$

The term Φ_{AF} on the right-hand side of (6a) is the AF error in the three-factored ADI scheme given by (4). If we want to reduce the AF error in (4), then we must add the term Φ_{AF}, or part thereof, back to the right-hand side of (4).

The Fourier method of stability analysis shows that when second-order-accurate central-differencing formulas are used for the spatial derivatives, the three-factored ADI scheme given by (4) is unconditionally unstable [11]. In contrast, the scheme given by (2) is unconditionally stable when used with the same second-order-accurate central-differencing formulas. This implies that the AF error destabilized the scheme. This also implies that if the AF errors in (4) can be reduced or eliminated (e.g., by adding Φ_{AF}), then the three-factored ADI scheme can be stabilized.

Here it is emphasized that it is unnecessary to add all of the terms in Φ_{AF} given by (6b) to (4) in order to stabilize the three-factored ADI scheme. To illustrate this point, we consider another way of factoring (2) that yields the following unconditionally stable scheme when second-order-accurate central-differencing formulas are used for the spatial derivatives:

(7) $$\left[1 + \frac{\gamma \Delta t}{1+\theta}(c_x D_x + c_y D_y)\right]\left(1 + \frac{\gamma \Delta t}{1+\theta} c_z D_z\right) \Delta u^{n+1}_{ijk} = \mathrm{RHS}_{ijk},$$

which can be split as

(8a) $$\left[1 + \frac{\gamma \Delta t}{1+\theta}(c_x D_x + c_y D_y)\right] \Delta u^{**}_{ijk} = \mathrm{RHS}_{ijk},$$

$$\text{(8b)}\qquad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_zD_z)\right]\Delta u_{ijk}^{n+1}=\Delta u_{ijk}^{**}.$$

Equation (8a) can be rewritten as

$$\text{(9a)}\qquad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)\right]\Delta u_{ijk}^{*}=\text{RHS}_{ijk}+\left(\frac{\gamma\Delta t}{1+\theta}\right)^2(c_xD_x)(c_yD_y)\Delta u_{ijk}^{**},$$

$$\text{(9b)}\qquad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_yD_y)\right]\Delta u_{ijk}^{**}=\Delta u_{ijk}^{*},$$

so that (9a), (9b), and (8b) appear as a three-factored scheme and can be compared with the three-factored ADI scheme given by (5). If these equations are compared, it can be seen that to stabilize (4) it is necessary to add only the last term in (9a) to the right-hand side of (4). Adding this term would effectively convert the three-factored scheme given by (4) to the two-factored scheme given by (7).

3. AF-error-reduction techniques. This section presents three AF-error-reduction methods for reducing or eliminating AF errors in the three-factored ADI scheme given by (4). The first technique is designed to stabilize the three-factored ADI scheme by eliminating some, but not all, AF errors. The remaining two techniques are designed to eliminate all AF errors.

3.1. Technique 1: Stabilize the three-factored ADI scheme. As mentioned in §2, one way to stabilize the three-factored ADI scheme given by (4) is to convert it to the two-factored scheme given by (7). This can be achieved by adding

$$\left(\frac{\gamma\Delta t}{1+\theta}\right)^2(c_xD_x)(c_yD_y)\Delta u_{ijk}^{**}$$

to the right-hand side of (4) or, equivalently, to the right-hand side of (5a). The resulting scheme, given by (9a), (9b), and (8b), is repeated below for convenience:

$$\text{(10a)}\qquad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)\right]\Delta u_{ijk}^{*}=\text{RHS}_{ijk}+\left(\frac{\gamma\Delta t}{1+\theta}\right)^2(c_xD_x)(c_yD_y)\Delta u_{ijk}^{**},$$

$$\text{(10b)}\qquad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_yD_y)\right]\Delta u_{ijk}^{**}=\Delta u_{ijk}^{*},$$

$$\text{(10c)}\qquad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_zD_z)\right]\Delta u_{ijk}^{n+1}=\Delta u_{ijk}^{**}.$$

Note that the term that was added to (5a) to obtain (10a) is a cross derivative that couples together all grid points in a constant-z plane, and since it involves Δu^{**} it also couples (10b) to (10a). Thus in order to obtain an economical AF-error-reduction method, the cross-derivative term must be eliminated and (10a) and (10b) must be decoupled.

The cross-derivative term in (10a) is eliminated by rewriting (10b) as

$$\text{(11)}\qquad \frac{\gamma\Delta t}{1+\theta}(c_yD_y)\Delta u_{ijk}^{**}=\Delta u_{ijk}^{*}-\Delta u_{ijk}^{**}$$

and by substituting it into (10a) to give

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^{*}_{ijk} = \mathrm{RHS}_{ijk}+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u^{*}_{ijk}-\Delta u^{**}_{ijk}). \tag{12}$$

Equation (12) shows the cross derivative eliminated, but it also shows that (12) (which now replaces (10a)) is still coupled to (10b) since Δu^* and Δu^{**} appear in both.

To decouple (12) from (10b) we suggest the following iterative scheme:

Predictor Step

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^{*\prime}_{ijk} = \mathrm{RHS}_{ijk}, \tag{13a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u^{**\prime}_{ijk} = \Delta u^{*\prime}_{ijk}. \tag{13b}$$

Corrector Step

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^{*\prime\prime}_{ijk} = \mathrm{RHS}_{ijk}+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u^{*\prime}_{ijk}-\Delta u^{**\prime}_{ijk}), \tag{14a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u^{**\prime\prime}_{ijk} = \Delta^{*\prime\prime}_{ijk}, \tag{14b}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u^{n+1}_{ijk} = \Delta u^{**\prime\prime}_{ijk}. \tag{14c}$$

Depending on the problem and the time-step size, the corrector step given by (14a) and (14b) may need to be iterated a few times before (14c) is solved. Iterating (14a) and (14b) permits $\Delta u^{*\prime}$ and $\Delta u^{**\prime}$ to be made as close as desired to $\Delta u^{*\prime\prime}$ and $\Delta u^{**\prime\prime}$, respectively. In the limit, when $\Delta u^{*\prime} = \Delta u^{*\prime\prime}$ and $\Delta u^{**\prime} = \Delta u^{**\prime\prime}$, (14a) becomes

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^{*\prime\prime}_{ijk} = \mathrm{RHS}_{ijk}+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u^{*\prime\prime}_{ijk}-\Delta u^{**\prime\prime}_{ijk}).$$

If (14b) is substituted for $\Delta u^{*\prime\prime}$, the above equation becomes

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u^{**\prime\prime}_{ijk} = \mathrm{RHS}_{ijk}+\left(\frac{\gamma\Delta t}{1+\theta}\right)^2(c_xD_x)(c_yD_y)\Delta u^{**\prime\prime}_{ijk},$$

which is identical to

$$\left[1+\frac{\gamma\Delta t}{1+\theta}(c_xD_x+c_yD_y)\right]\Delta u^{**\prime\prime}_{ijk} = \mathrm{RHS}_{ijk}.$$

Thus in the limit the AF-error-reduction scheme given by (13) and (14) can be used to convert the unconditionally unstable three-factored ADI scheme given by (4) to the unconditionally stable two-factored scheme given by (7).

In §4 it will be shown that when central-differencing formulas are used for all spatial derivatives, the iteration process given by (14a) and (14b) is convergent for all values of Δt. This iteration process can, therefore, be said to be *unconditionally convergent*. It is also shown in §4 that the overall algorithm given by (13) and (14) can be made conditionally stable and that its stability characteristics improve as the number of iterations of (14a) and (14b) is increased.

3.2. Technique 2: Reduce all AF errors in the three-factored ADI scheme. The AF-error-reduction method presented in the previous subsection eliminated only AF errors associated with one out of the two factorizations in the three-factored ADI scheme. This subsection presents a method of reducing or eliminating AF errors associated with all factorizations in the three-factored ADI scheme.

Consider the exactly factored scheme given by (6), which can be split as follows:

(15a)
$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^*_{ijk} = \mathrm{RHS}_{ijk} + \left(\frac{\gamma\Delta t}{1+\theta}\right)^2[(c_xD_x)(c_yD_y)+(c_xD_x)(c_zD_z)+(c_yD_y)(c_zD_z)]\Delta u^{n+1}_{ijk} + \left(\frac{\gamma\Delta t}{1+\theta}\right)^3[(c_xD_x)(c_yD_y)(c_zD_z)]\Delta u^{n+1}_{ijk},$$

(15b)
$$\left[1+\frac{\gamma\Delta t}{1+\theta}(c_yD_y)\right]\Delta u^{**}_{ijk} = \Delta u^*_{ijk},$$

(15c)
$$\left[1+\frac{\gamma\Delta t}{1+\theta}(c_zD_z)\right]\Delta u^{n+1}_{ijk} = \Delta u^{**}_{ijk}.$$

To eliminate the cross-derivative terms in (15a), (15b), and (15c) are rewritten as

(16a)
$$\frac{\gamma\Delta t}{1+\theta}(c_zD_z)\Delta u^{n+1}_{ijk} = \Delta u^{**}_{ijk} - \Delta u^{n+1}_{ijk},$$

(16b)
$$\frac{\gamma\Delta t}{1+\theta}(c_yD_y)\Delta u^{**}_{ijk} = \Delta u^{*}_{ijk} - \Delta u^{**}_{ijk}.$$

Inserting (16a) and (16b) into (15a) gives

(17)
$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^*_{ijk} = \mathrm{RHS}_{ijk} + \frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u^*_{ijk} - \Delta u^{n+1}_{ijk}) + \frac{\gamma\Delta t}{1+\theta}(c_yD_y)(\Delta u^{**}_{ijk} - \Delta u^{n+1}_{ijk}).$$

This suggests the following method for eliminating all AF errors.

Predictor Step

(18a)
$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u^{*\prime}_{ijk} = \mathrm{RHS}_{ijk},$$

(18b)
$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u^{**\prime}_{ijk} = \Delta u^{*\prime}_{ijk},$$

(18c)
$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u^{(n+1)\prime}_{ijk} = \Delta u^{**\prime}_{ijk}.$$

Corrector Step

$$\text{(19a)}\quad \left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u_{ijk}^{*\prime\prime} = \mathrm{RHS}_{ijk} + \frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u_{ijk}^{*\prime} - \Delta u_{ijk}^{(n+1)'}) + \frac{\gamma\Delta t}{1+\theta}(c_yD_y)(\Delta u_{ijk}^{**\prime} - \Delta u_{ijk}^{(n+1)'}),$$

$$\text{(19b)}\quad \left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u_{ijk}^{**\prime\prime} = \Delta u_{ijk}^{*\prime\prime},$$

$$\text{(19c)}\quad \left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u_{ijk}^{(n+1)''} = \Delta u_{ijk}^{**\prime\prime},$$

where the corrector step may have to be repeated more than once.

When second-order-accurate central-differencing formulas are used for the spatial derivatives, it can be shown that the iteration process given by (19) is conditionally convergent, i.e., it converges only if the time-step Δt is smaller than some critical value. For ensuring convergence the Δt used must satisfy the following condition:

$$\text{(20)}\quad 2a_xa_ya_z(a_x+a_y+a_z) - (a_x^2+a_y^2+a_z^2) \leqq 1,$$

where

$$\text{(21)}\quad \begin{gathered} a_x = \frac{\gamma}{1+\theta}v_x, \quad a_y = \frac{\gamma}{1+\theta}v_y, \quad a_z = \frac{\gamma}{1+\theta}v_z, \\ v_x = \frac{|c_x|\Delta t}{\Delta x}, \quad v_y = \frac{|c_y|\Delta t}{\Delta y}, \quad v_z = \frac{|c_z|\Delta t}{\Delta z}. \end{gathered}$$

If $a_x = a_y = a_z = a$, then the convergence condition becomes

$$\text{(22)}\quad a \leqq 0.85,$$

or

$$\text{(23)}\quad \Delta t \leqq 0.85\frac{1+\theta}{\gamma}\left(\frac{\Delta x}{c_x}\right).$$

The convergence criterion given above is quite restrictive. Thus the AF-error-reduction method given by (18) and (19) is uneconomical. Below, a more economical method for reducing all AF errors in the three-factored ADI scheme is presented.

3.3. Technique 3: Reduce all AF errors in the ADI three-factored scheme. The AF-error-reduction method presented in §3.2 was based on the exact factoring of (2) given by (6). This subsection presents an AF-error-reduction scheme that is based on a different exact factoring of (2).

A different exact factoring of (2) proceeds as follows. First, factor and split (2) as

$$\text{(24a)}\quad \left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u_{ijk}^{*} = \mathrm{RHS}_{ijk} + \left(\frac{\gamma\Delta t}{1+\theta}\right)^2 (c_xD_x)(c_yD_y+c_zD_z)\Delta u_{ijk}^{n+1},$$

$$\text{(24b)}\quad \left[1+\frac{\gamma\Delta t}{1+\theta}(c_yD_y+c_zD_z)\right]\Delta u_{ijk}^{n+1} = \Delta u_{ijk}^{*}.$$

Then, factor and split (24b) as

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u_{ijk}^{**}=\Delta u_{ijk}^{*}+\left(\frac{\gamma\Delta t}{1+\theta}\right)^2(c_yD_y)(c_zD_z)\Delta u_{ijk}^{n+1}, \tag{25a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u_{ijk}^{n+1}=\Delta u_{ijk}^{**}. \tag{25b}$$

By eliminating the cross-derivative terms in the same manner as before, we obtain

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u_{ijk}^{*}=\mathrm{RHS}_{ijk}+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u_{ijk}^{*}-\Delta u_{ijk}^{n+1}), \tag{26a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u_{ijk}^{**}=\Delta u_{ijk}^{*}+\frac{\gamma\Delta t}{1+\theta}(c_yD_y)(\Delta u_{ijk}^{**}-\Delta u_{ijk}^{n+1}), \tag{26b}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u_{ijk}^{n+1}=\Delta u_{ijk}^{**}. \tag{26c}$$

Equations (26a), (26b), and (26c) suggest the following AF-error-reduction method:

Predictor Step

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u_{ijk}^{*\prime}=\mathrm{RHS}_{ijk}, \tag{27a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u_{ijk}^{**\prime}=\Delta u_{ijk}^{*\prime}, \tag{27b}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u_{ijk}^{(n+1)\prime}=\Delta u_{ijk}^{**\prime}. \tag{27c}$$

Corrector Step

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\Delta u_{ijk}^{*\prime\prime}=\mathrm{RHS}_{ijk}+\frac{\gamma\Delta t}{1+\theta}(c_xD_x)(\Delta u_{ijk}^{*\prime}-\Delta u_{ijk}^{(n+1)\prime}), \tag{28a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\Delta u_{ijk}^{**\prime\prime}=\Delta_{ijk}^{*\prime\prime}+\frac{\gamma\Delta t}{1+\theta}(c_yD_y)(\Delta u_{ijk}^{**\prime}-\Delta u_{ijk}^{(n+1)\prime}), \tag{28b}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_zD_z\right)\Delta u_{ijk}^{(n+1)\prime\prime}=\Delta u_{ijk}^{**\prime\prime}. \tag{28c}$$

Note that in the AF-error-reduction method given by (18) and (19), the terms corresponding to the AF error were all put in the first factor (19a). In the current AF-error-reduction method the AF error terms are split between the first and the second factors ((28a) and (28b)).

As in the previous AF-error-reduction methods, the corrector step given by (28) may have to be repeated several times. Equations (28) can be iterated in two ways. The first way is to iterate (28a) to (28c) within a single iteration loop until a converged solution for Δu^{n+1} is obtained. The convergence criteria for this algorithm is complicated and difficult to analyze.

The second way to iterate (28) is to use two nested loops. Within the inner loop, (28b) and (28c) are iterated (for a fixed Δu^*) until a converged (interim) solution for Δu^{n+1} is achieved. Then, in the outer loop, (28a) is used to compute a new Δu^*, based on the interim solution for Δu^{n+1}, before the inner loop is executed again. This algorithm can be shown to be unconditionally convergent, provided that the inner loop of the iteration scheme is carried out until a fully converged (interim) solution is obtained. This result follows from the observation that the inner loop is simply an unconditionally convergent two-factored scheme similar to that given in §3.1. If, for each of the steps in the outer iteration loop, the inner loop is carried out to full convergence, then the outer iteration loop also corresponds to a two-factored scheme that can be shown to be unconditionally convergent. With the above arguments it can be deduced that the iteration scheme given by (28) is unconditionally convergent.

In this section, three AF-error-reduction methods were presented. Although all three methods were presented in the framework of the three-factored ADI scheme, these methods can easily be extended and applied to other schemes constructed by using AF.

4. Analysis. In this section, the AF-error-reduction method given by (13) and (14) is analyzed for its convergence and stability properties. The other two AF-error-reduction methods can be analyzed in a similar way.

For the AF-error-reduction method given by (13) and (14) to be useful, the iteration process given by (14) must be convergent and the overall algorithm given by (13) and (14) must be stable. Below, the convergence property is analyzed first, and then the stability property is analyzed.

To analyze the convergence property of the iteration process given by (14), we note that if the iteration process converges, then (14a) must approach (12) and (14b) must approach (10b) as the number of iterations increases. Subtracting (14a) from (12) and subtracting (14b) from (10b) gives

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\epsilon_{ijk}^m = \frac{\gamma\Delta t}{1+\theta}(c_xD_x)[\epsilon_{ijk}^{m-1}-\delta_{ijk}^{m-1}], \tag{29a}$$

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\delta_{ijk}^m = \epsilon_{ijk}^m, \tag{29b}$$

where ϵ_{ijk}^m is the error in Δu^* after the mth iteration and δ_{ijk}^m is the error in Δu^{**} after the mth iteration, i.e.,

$$\epsilon_{ijk}^m = \Delta u_{ijk}^* - \Delta u_{ijk}^{*\prime\prime}, \tag{30a}$$

$$\delta_{ijk}^m = \Delta u_{ijk}^{**} - \Delta u_{ijk}^{**\prime\prime}. \tag{30b}$$

Substituting (29b) into (29a) gives

$$\left(1+\frac{\gamma\Delta t}{1+\theta}c_xD_x\right)\left(1+\frac{\gamma\Delta t}{1+\theta}c_yD_y\right)\delta_{ijk}^m = \left(\frac{\gamma\Delta t}{1+\theta}\right)^2(c_xD_x)(c_yD_y)\delta_{ijk}^{m-1}. \tag{31}$$

Now, suppose that the error δ^m_{ijk} has the following form:

$$\delta^m_{ijk} = D^m \exp[I(k_x x_i + k_y y_j + k_z z_k)], \tag{32}$$

where $I = \sqrt{-1}$; k_x, k_y, and k_z are the wave numbers in the x, y, and z, directions, respectively; and x_i, y_j, and z_k are the coordinates of the grid point at (i, j, k). Since the spatial derivatives are approximated by central-difference operators, (32) implies

$$D_x\delta^m_{ijk} = I\delta^m_{ijk}\frac{\sin(k_x\Delta x)}{\Delta x}, \tag{33a}$$

$$D_y\delta^m_{ijk} = I\delta^m_{ijk}\frac{\sin(k_y\Delta y)}{\Delta y}. \tag{33b}$$

Substituting (32) and (33) into (31) and dividing through by

$$\exp[I(k_x x_i + k_y y_j + k_z z_k)]$$

gives

$$\begin{aligned} &D^m(1 + Ia_x\sin(k_x\Delta x))(1 + Ia_y\sin(k_y\Delta y)) \\ &= -D^{m-1}a_x a_y \sin(k_x\Delta x)\sin(k_y\Delta_y), \end{aligned} \tag{34}$$

where

$$a_x = \frac{\gamma}{1+\theta}v_x, \quad a_y = \frac{\gamma}{1+\theta}v_y, \quad v_x = \frac{c_x\Delta t}{\Delta x}, \quad v_y = \frac{c_y\Delta t}{\Delta y}. \tag{35}$$

Equation (34) can be rearranged to give the following amplification factor:

$$G = \frac{D^m}{D^{m-1}} = \frac{-a_x a_y\sin(k_x\Delta x)\sin(k_y\Delta y)}{(1 + Ia_x\sin(k_x\Delta x))(1 + Ia_y\sin(k_y\Delta y))} \tag{36}$$

or

$$|G|^2 = \frac{a_x^2 a_y^2\sin^2(k_x\Delta x)\sin^2(k_y\Delta y)}{1 + a_x^2\sin^2(k_x\Delta x) + a_y^2\sin^2(k_y\Delta y) + a_x^2 a_y^2\sin^2(k_x\Delta x)\sin^2(k_y\Delta y)}. \tag{37}$$

From (37) it is clear that $|G|^2$ is less than unity for all wave numbers. This indicates that the iteration process given by (14a) and (14b) is unconditionally convergent.

With the unconditional convergence of the iteration process shown, the stability of the overall algorithm given by (13) and (14) is now analyzed. For simplicity, the Euler implicit time-differencing formula is used, so that $\gamma = 1$ and $\theta = 0$. For this case, (13) and (14), along with (3) can be written as

$$\begin{aligned} &(1 + \Delta t c_x D_x)(1 + \Delta t c_y D_y)(1 + \Delta t c_z D_z)\Delta u^{n+1}_{ijk} \\ &= -\Delta t(c_x D_x + c_y D_y + c_z D_z)u^n_{ijk} \\ &\quad + \Delta t^2(c_x D_x)(c_y D_y)\Delta u^{**\prime}_{ijk}. \end{aligned} \tag{38}$$

Now we introduce a parameter α defined by

$$(c_x D_x)(c_y D_y)\Delta u^{**\prime} = \alpha(c_x D_x)(c_y D_y)\Delta u^{**}. \tag{39}$$

Note that as the number of iterations of (14a) and (14b) is increased, the value of α approaches unity and $\Delta u^{**\prime}$ approaches Δu^{**}. If we use (10c) and the parameter α defined by (39), equation (38) becomes

$$\begin{aligned}&\{1+\Delta t(c_xD_x+c_yD_y+c_zD_z)+\Delta t^2[(1-\alpha)(c_xD_x)(c_yD_y)\\&\quad+(c_xD_x)(c_zD_z)+(c_yD_y)(c_zD_z)]\\&\quad+(1-\alpha)\Delta t^3(c_xD_x)(c_yD_y)(c_zD_z)\}\Delta u_{ijk}^{n+1}\\&\quad=-\Delta t(c_xD_x+c_yD_y+c_zD_z)u_{ijk}^n.\end{aligned} \tag{40}$$

Application of the Fourier method of stability analysis to (40) readily yields an amplification factor with the following modulus:

$$|G|^2=\frac{(1-\xi)^2-\eta(\zeta-\eta)}{(1-\xi)^2+(\zeta-\eta)^2}, \tag{41}$$

where

$$\begin{aligned}\xi=&(1-\alpha)v_xv_y\sin(k_x\Delta x)\sin(k_y\Delta y)\\&+v_xv_z\sin(k_x\Delta x)\sin(k_z\Delta z)\\&+v_yv_z\sin(k_y\Delta y)\sin(k_z\Delta z),\end{aligned} \tag{42a}$$

$$\eta=(1-\alpha)v_xv_yv_z\sin(k_x\Delta x)\sin(k_y\Delta y)\sin(k_z\Delta z), \tag{42b}$$

$$\zeta=v_x\sin(k_x\Delta x)+v_y\sin(k_y\Delta y)+v_z\sin(k_z\Delta z), \tag{42c}$$

$$v_x=\frac{c_x\Delta t}{\Delta x},\quad v_y=\frac{c_y\Delta t}{\Delta y},\quad v_z=\frac{c_z\Delta t}{\Delta z}. \tag{43}$$

For numerical stability, the modulus given by (41) must be less than or equal to unity, i.e., $|G|\leqq 1$. From (42), it is clear that if $\alpha=1$, then $|G|\leqq 1$ for all wave numbers. This indicates that if the iteration of (14a) and (14b) is carried out until a converged solution is obtained, then the method given by (13) and (14) is unconditionally stable. If only a finite number of iterations of (14a) and (14b) are carried out, then α will be less than unity, and this results in a conditionally stable algorithm. Another way to state this is that, for a given Δt, there is a certain number of iterations that need to be carried out before the algorithm becomes stable. The number of iterations needed to stabilize the algorithm is problem dependent.

Note that for the original three-factored ADI scheme given by (4), $\alpha=0$. For that case Ying [9] has shown that the maximum $|G|$ is greater than unity, indicating unconditional instability.

5. Numerical experiments. In this section the usefulness of the AF-error-reduction methods developed in this study are demonstrated by applying them to a test problem. The test problem selected is to obtain a steady-state solution of (1) by using the three-factored ADI scheme. Since the three-factored ADI scheme applied to (1) is unconditionally unstable when central-differencing formulas are used to approximate the spatial derivatives, this problem is a stringent test of the AF-error-reduction methods presented.

5.1. Test problem and solution algorithm. In this subsection the test problem is formulated and the solution algorithms with the AF-error-reduction methods applied to the test problem are summarized.

All exact steady-state solutions to (1) must satisfy

$$\mathbf{c} \cdot \nabla u = 0,$$

where $\mathbf{c} = (c_x, c_y, c_z)^T$. This implies that steady-state solutions must be constant on lines parallel to the velocity vector $\mathbf{c}$. Thus steady-state solutions of (1) are functions of the vector $\mathbf{X} = (\xi, \eta, \zeta)^T$ given by

$$\mathbf{X} = \mathbf{r} - (\mathbf{r} \cdot \mathbf{s})\mathbf{s}, \tag{44}$$

where

$$\mathbf{r} = \begin{bmatrix} x \\ y \\ x \end{bmatrix}, \qquad \mathbf{s} = \frac{\mathbf{c}}{|\mathbf{c}|}.$$

The components of $\mathbf{X}$ can be written as

$$\xi = x - c_x \left(\frac{c_x x + c_y y + c_z z}{c_x^2 + c_y^2 + c_z^2} \right), \tag{45a}$$

$$\eta = y - c_y \left(\frac{c_x x + c_y y + c_z z}{c_x^2 + c_y^2 + c_z^2} \right), \tag{45b}$$

$$\zeta = z - c_z \left(\frac{c_x x + c_y y + c_z z}{c_x^2 + c_y^2 + c_z^2} \right), \tag{45c}$$

Note that $\mathbf{X}$ is constant on lines parallel to the vector $\mathbf{c}$ and that this constant is zero for the line passing through the origin.

With (44) and (45) a test problem that has a known steady-state solution to (1) can be constructed. The test problem selected is as follows: the spatial domain is taken to be a unit cube, i.e.,

$$0 \leqq x \leqq 1, \quad 0 \leqq y \leqq 1, \quad 0 \leqq z \leqq 1. \tag{46}$$

The velocity vector $\mathbf{c}$ is taken as

$$\mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \tag{47}$$

so that it has nonzero component in all three coordinate directions. With this choice of $\mathbf{c}$, boundaries at $x = 0$, $y = 0$, and $z = 0$ are inflow boundaries and the boundaries at $x = 1$, $y = 1$, and $z = 1$ are outflow boundaries. The boundary conditions imposed at the inflow boundaries are those that yield the following steady-state solution to (1):

$$u_s(\xi, \eta, \zeta) = \cos(2\pi\xi) + \cos(2\pi\eta) + \cos(2\pi\zeta). \tag{48}$$

Note that the steady-state solution given by (48) has nonzero first-order derivatives in all three coordinate directions. The initial condition used for (1) is given by

$$u(t=0,x,y,z) = u_s(\xi,\eta,\zeta) + 0.51\sin\left(\frac{\pi x}{10}\right)\sin\left(\frac{\pi y}{20}\right)\sin\left(\frac{\pi z}{30}\right), \tag{49}$$

which differs from the steady-state solution given by (48) by the sinusoidal disturbance.

To obtain solutions to (1) constrained by (46)–(49) by using an FD or FV method, such as those described in the previous sections, the domain must be discretized and additional boundary conditions (known as numerical boundary conditions) must be provided at the outflow boundaries at $x = 1$, $y = 1$, and $z = 1$.

Here the temporaral domain was replaced by equally incremented time steps. Time-step sizes of 0.001, 0.002, 0.008, 0.01, 0.015, 0.02, 0.03, 0.04, and 0.06 s were investigated. The spatial domain given by (46) was replaced by (IL $= 59$) $\times$ (JL $= 53$) $\times$ (KL $= 47$) equally spaced grid points, where IL, JL, and KL are the number of grid points in the x, y, and z directions, respectively. Note that with this choice of IL, JL, and KL, the grid spacings in the x, y, and z directions are different.

The numerical boundary conditions used at the outflow boundaries are

$$u_{\mathrm{IL},j,k} = 2u_{\mathrm{IL}-1,j,k} - u_{\mathrm{IL}-2,j,k}, \tag{50a}$$

$$u_{i,\mathrm{JL},k} = 2u_{i,\mathrm{JL}-1,k} - u_{i,\mathrm{JL}-2,k}, \tag{50b}$$

$$u_{i,j,\mathrm{KL}} = 2u_{i,j,\mathrm{KL}-1} - u_{i,j,\mathrm{KL}-2}. \tag{50c}$$

Since the three-factored ADI scheme and the AF-error-reduction methods are implicit, all boundary conditions, including those given above, were implemented implicitly as follows:

$$\Delta u^{n+1}_{1,j,k} = \Delta u^{n+1}_{i,1,k} = \Delta u^{n+1}_{i,j,1} = 0, \tag{51a}$$

$$\Delta u^{n+1}_{\mathrm{IL},j,k} = 2\Delta u^{n+1}_{\mathrm{IL}-1,j,k} - \Delta u^{n+1}_{\mathrm{IL}-2,j,k}, \tag{51b}$$

$$\Delta u^{n+1}_{i,\mathrm{JL},k} = 2\Delta u^{n+1}_{i,\mathrm{JL}-1,k} - \Delta u^{n+1}_{i,\mathrm{JL}-2,k}, \tag{51c}$$

$$\Delta u^{n+1}_{i,j,\mathrm{KL}} = 2\Delta u^{n+1}_{i,j,\mathrm{KL}-1} - \Delta u^{n+1}_{i,j,\mathrm{KL}-2}. \tag{51d}$$

Numerical solutions to (1), along with (46)–(50), were obtained by using the following three methods:

(i) The original three-factored ADI scheme given by (5).

(ii) The three-factored ADI scheme with AF-error-reduction Technique 1 given by (13) and (14) (henceforth referred to as AFRT-1), which stabilizes the three-factored scheme by converting it to a two-factored scheme.

(iii) The three-factored ADI scheme with AF-error-reduction Technique 3 given by (27) and (28) (henceforth referred to as AFRT-3), which stabilizes the three-factored scheme by reducing all AF errors.

Note that AF-error-reduction Technique 2, presented in §3.2, was not tested because of the stringent criteria that must be satisfied in order for the iteration process involved to converge.

To illustrate that the AF-error-reduction methods developed in this study can easily be incorporated into existing codes based on two- or three-factored schemes, the algorithms for implementing the three methods listed above are given below.

Three-Factored ADI *Scheme*

```
specify initial conditions
for $n = 1$ to $N_{\mathrm{MAX}}$ do:
    compute $\Delta u^*$ (equation (5a))
    compute $\Delta u^{**}$ (equation (5b))
    compute $\Delta u^{n+1}$ (equation (5c))
    compute $\Delta u^{n+1}$ on boundaries (equations (51))
    $u^{n+1} \leftarrow u^n + \Delta u^{n+1}$
end do
STOP
```

AF-Error-Reduction Technique 1 (AFRT-1)

```
specify initial conditions
for $n = 1$ to $N_{\mathrm{MAX}}$ do:
    Predictor Step:
    compute $\Delta u^{*\prime}$ (equation (13a))
    compute $\Delta u^{**\prime}$ (equation (13b))
    Iterative Corrector Step:
    for $m = 1$ to $M_{\mathrm{MAX}}$ do:
        compute $\Delta u^{**\prime}$ on boundaries (equations (51))
        compute $\Delta u^{*\prime\prime}$ (equation (14a))
        compute $\Delta u^{**\prime\prime}$ (equation (14b))
        $\Delta u^{**\prime} \leftarrow \Delta u^{**\prime\prime}$
        $\Delta u^{*\prime} \leftarrow \Delta u^{*\prime\prime}$
    end do
    compute $\Delta u^{n+1}$ (equation (14c))
    compute $\Delta u^{n+1}$ on boundaries (equations (51))
    $u^{n+1} \leftarrow u^n + \Delta \quad u^{n+1}$
end do
STOP
```

AF-Error-Reduction Technique 3 (AFRT-3)

```
specify initial conditions
for $n = 1$ to $N_{\mathrm{MAX}}$ do:
    Predictor Step:
    compute $\Delta u^{*\prime}$ (equation (27a))
    compute $\Delta u^{**\prime}$ (equation (27b))
    compute $\Delta u^{(n+1)\prime}$ (equation (27c))
    compute $\Delta u^{(n+1)\prime}$ on boundaries (equations (51))
    Iterative Corrector Step:
    for $m = 1$ to $M_{\mathrm{MAX}}$ do:
        compute $\Delta u^{*\prime\prime}$ (equation (28a))
        for $l = 1$ to $L_{\mathrm{MAX}}$ do:
            compute $\Delta u^{**\prime}$ on boundaries (equations (51))
            compute $\Delta u^{**\prime\prime}$ (equation (28b))
```

```
                compute $\Delta u^{(n+1)''}$ (equation (28c))
                compute $\Delta u^{(n+1)''}$ on boundaries (equations (51))
                $\Delta u^{**'} \leftarrow \Delta u^{**''}$
                $\Delta u^{(n+1)'} \leftarrow \Delta u^{(n+1)''}$
            end do
        end do
        $u^{n+1} \leftarrow u^n + \Delta u^{n+1}$
    end do
    STOP
```

For the algorithms given above, N_{MAX} is the total number of time steps taken; M_{MAX} is the total number of corrector steps (i.e., the number of times the corrector step is iterated), and L_{MAX} is the total number of iterations performed within each corrector step (used only with AFRT-3). In this study L_{MAX} was always taken to be three times M_{MAX}. Note that the boundary conditions given by (51) were used not only for computing Δu^{n+1} but also for computing the intermediate variables Δu^* and Δu^{**}. This was found to be necessary in order to ensure convergence of the iteration processes involved.

5.2. Results. The AF-error-reduction techniques developed in this study were tested by applying them to the test problem described in §5.1 and by using the algorithms described there. In this subsection the results of these tests are presented.

The results of all numerical tests conducted are shown in Figs. 1–10. These figures show the L_2 norm of Δu^{n+1} plotted against time (Fig. 1) and against the number of time steps (Figs. 2–10).

Figure 1 shows the results obtained by using the ADI three-factored scheme without any AF-error-reduction technique. The results plotted are the convergence histories of the solutions obtained by using time-step sizes ranging from 0.001 to 0.01 s. As can be seen in the figure, converged solutions were obtained for all time-step sizes tested with this range. These converged solutions were obtained despite the prediction by the Fourier method of stability analysis, which states that the three-factored ADI scheme is unconditionally unstable. This apparent contradiction can be explained by the fact that the Fourier method does not consider the effects of finite-size domains and boundary conditions on numerical stability. In this study the boundary conditions used ($\Delta u = 0$ at inflow boundaries and upwind extrapolation at outflow boundaries) helped stabilize the three-factored ADI scheme. Note that for time-step sizes larger than 0.01 s the ADI three-factored scheme was found to be unstable.

Figures 2–10 show results obtained by using the AF-error-reduction techniques AFRT-1 and AFRT-3. Each figure shows the results obtained by using time-step size.

Figure 2 shows the results obtained by using time-step $\Delta t = 0 \cdot 001$ s. As can be seen, the AF-error-reduction techniques have no effect on the convergence rate except near the beginning of the computations. This indicates that for small time-step sizes AF errors are not significant and do not affect the solution.

Figures 3 and 4 show the results obtained by using $\Delta t = 0.008$ s for AFRT-1 and AFRT-3, respectively. From these figures it can be seen that the AF-error-reduction techniques do improve the convergence rate slightly. Figure 3 shows that for AFRT-1 one corrector step has the same effect on the convergence rate as two, four, or eight corrector steps. This indicates that for $\Delta t = 0.008$ s the iteration process used in AFRT-1 converges fully in only one corrector step. Comparison of Figs. 3 and 4 shows that

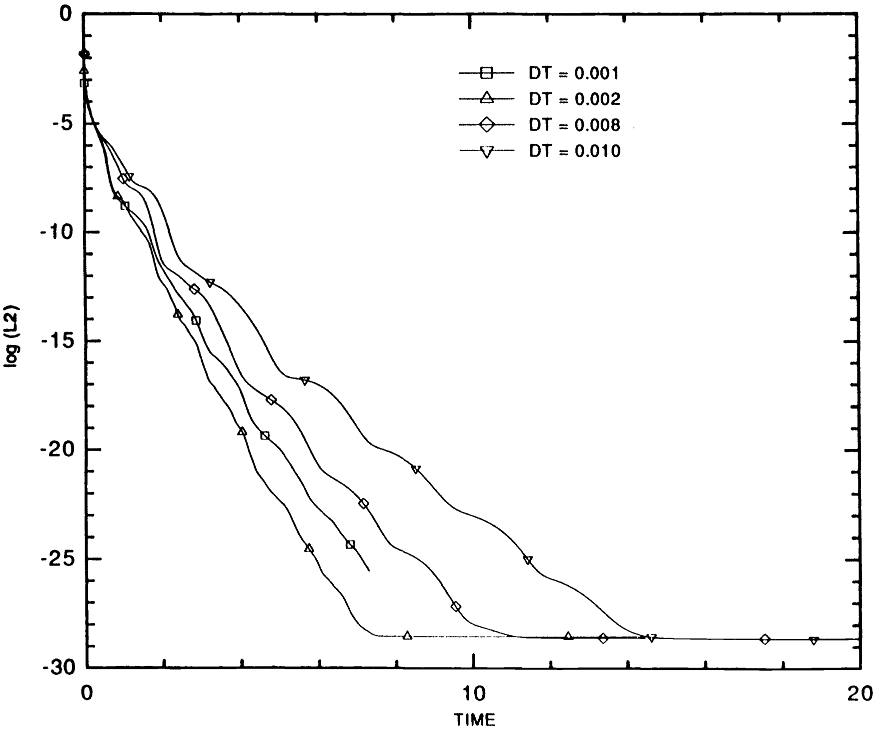


FIG. 1. L_2 *norm versus time for* ADI *three-factored scheme with* Δt *ranging from* 0.001 *to* 0.01 s.

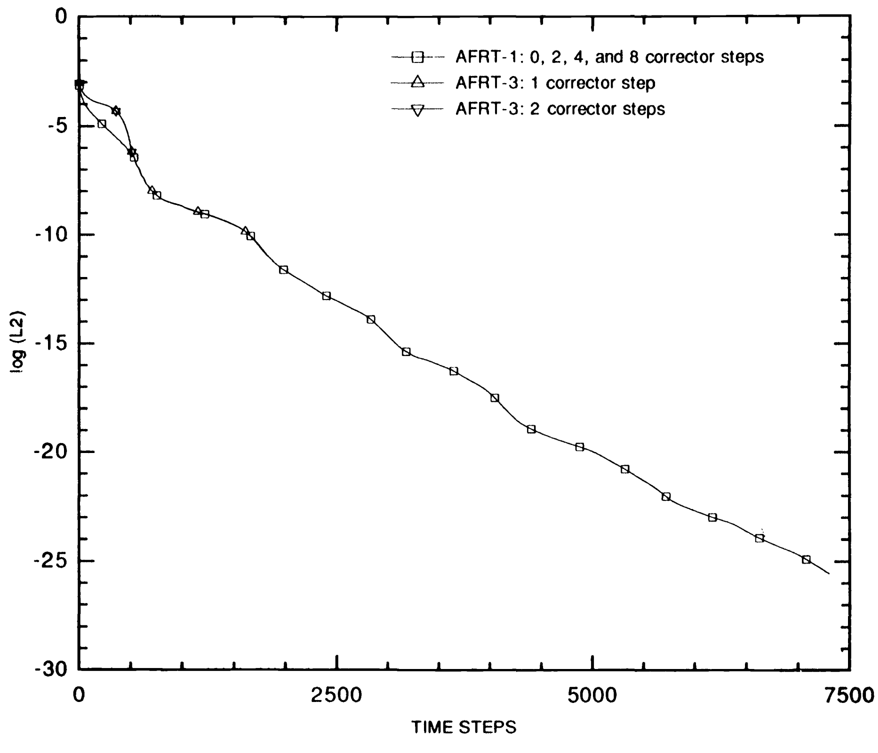


FIG. 2. L_2 *norm versus time steps for* AFRT-1 *and* AFRT-3 *with* $\Delta t = 0.001$ s.

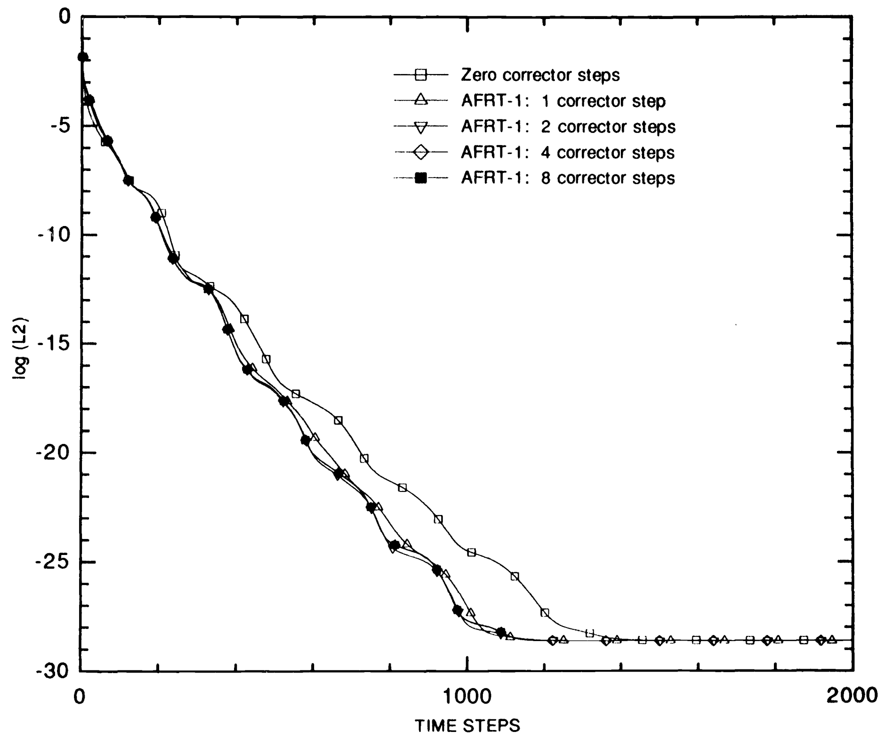


FIG. 3. L_2 *norm versus time steps for* AFRT-1 *with* $\Delta t = 0.008$ s.

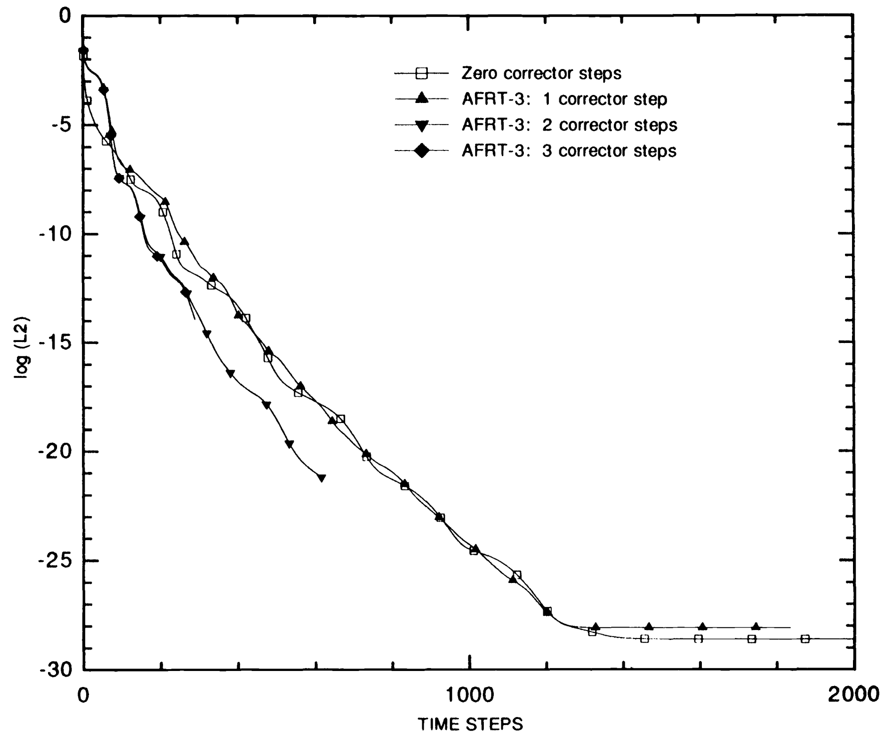


FIG. 4. L_2 *norm versus time steps for* AFRT-3 *with* $\Delta t = 0.008$ s.

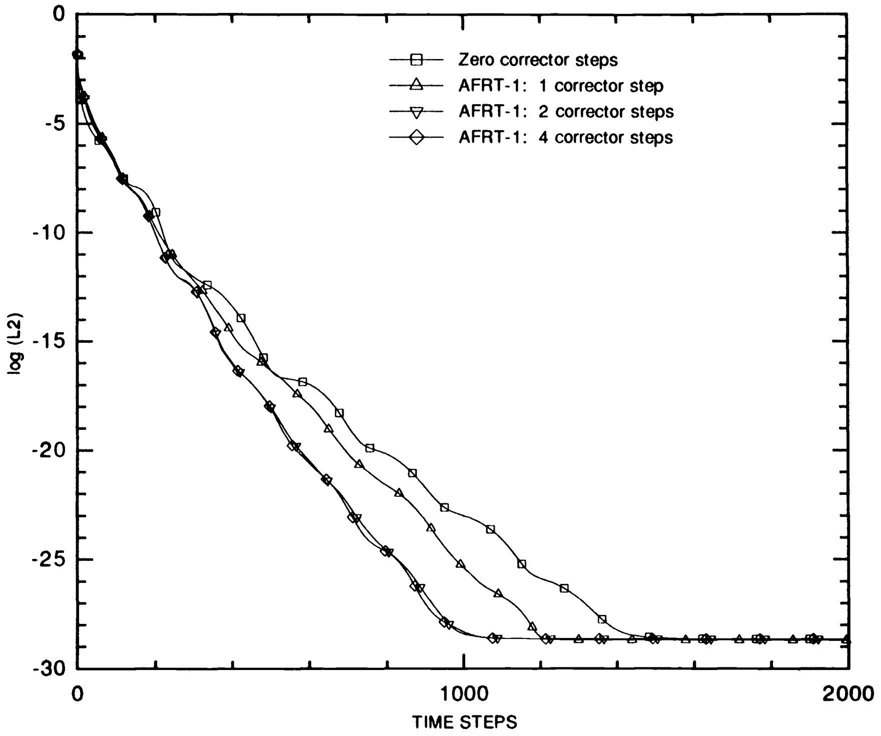


FIG. 5. L_2 *norm versus time steps for* AFRT-1 *with* $\Delta t = 0.01$ s.

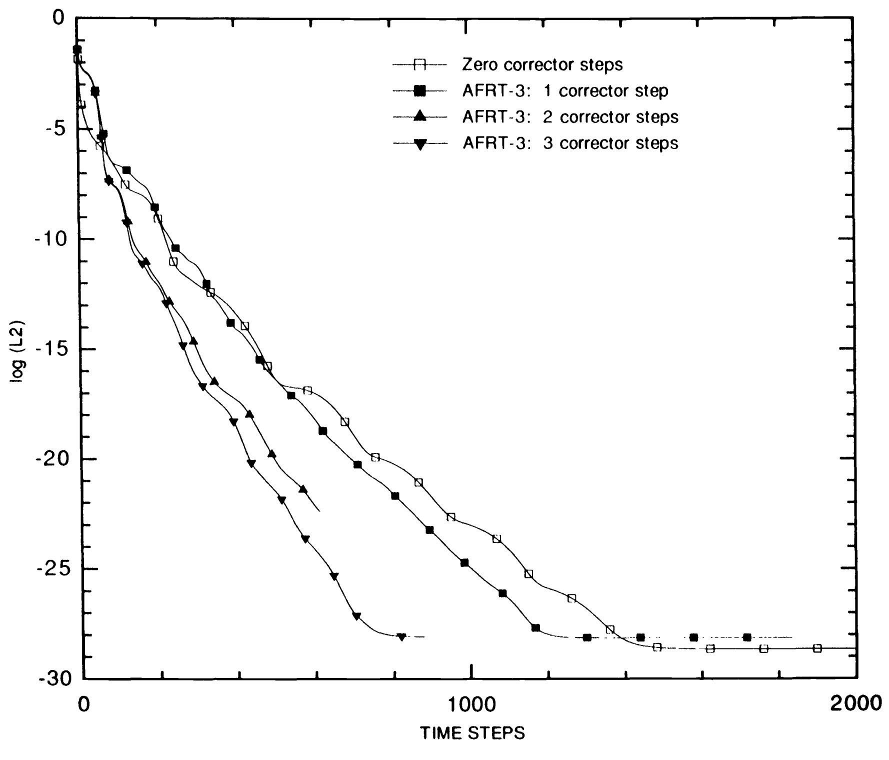


FIG. 6. L_2 *norm versus time steps for* AFRT-3 *with* $\Delta t = 0.01$ s.

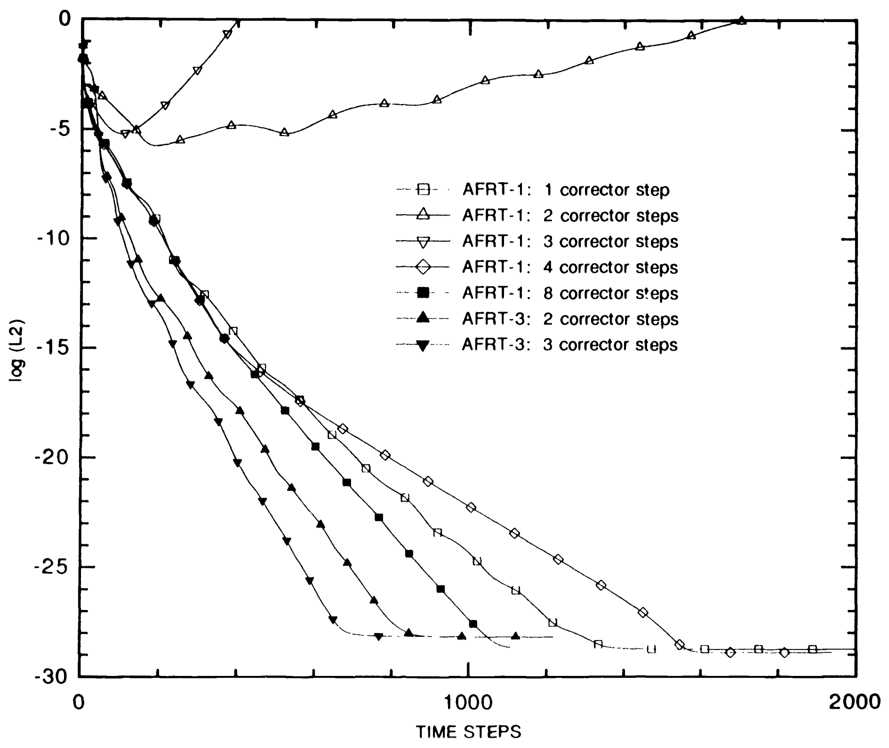


FIG. 7. L_2 *norm versus time steps for* AFRT-1 *and* AFRT-3 *with* $\Delta t = 0.015$ s.

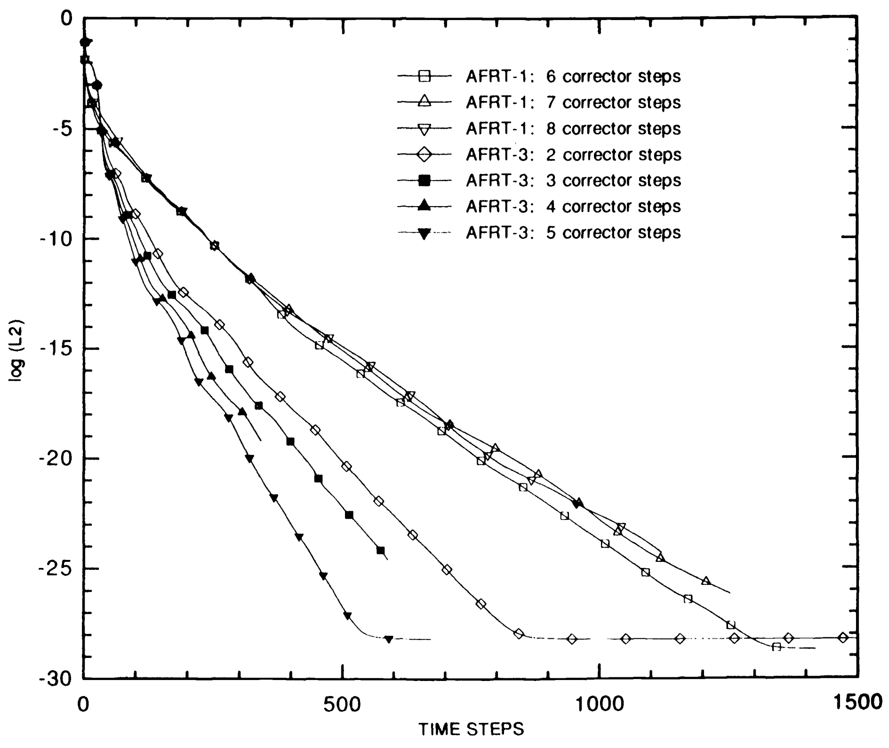


FIG. 8. L_2 *norm versus time steps for* AFRT-1 *and* AFRT-3 *with* $\Delta t = 0.02$ s.

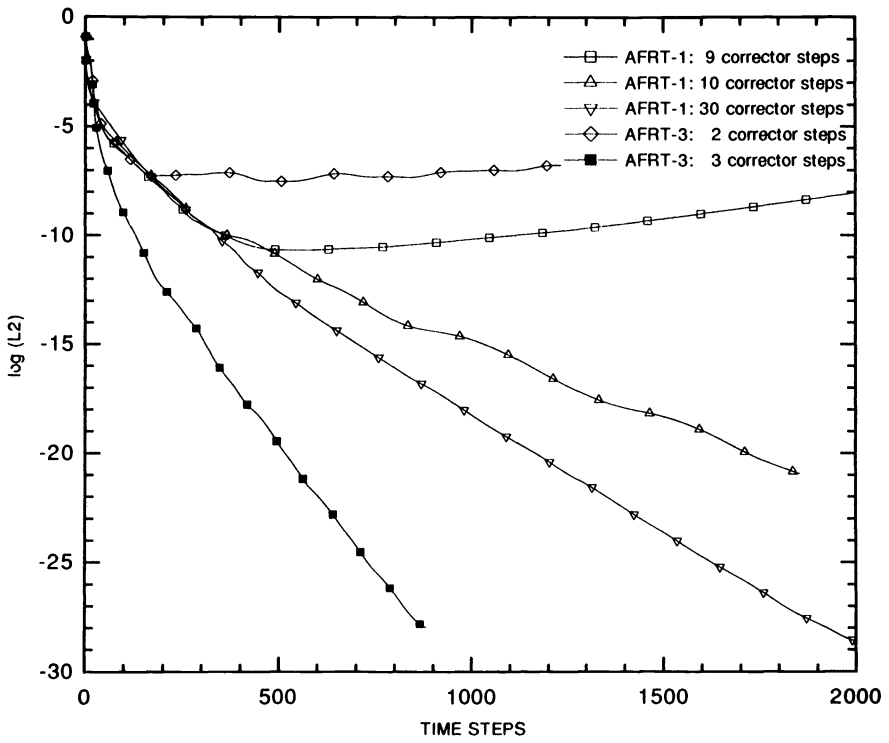


FIG. 9. *L_2 norm versus time steps for* AFRT-1 *and* AFRT-3 *with* $\Delta t = 0.03$ s.

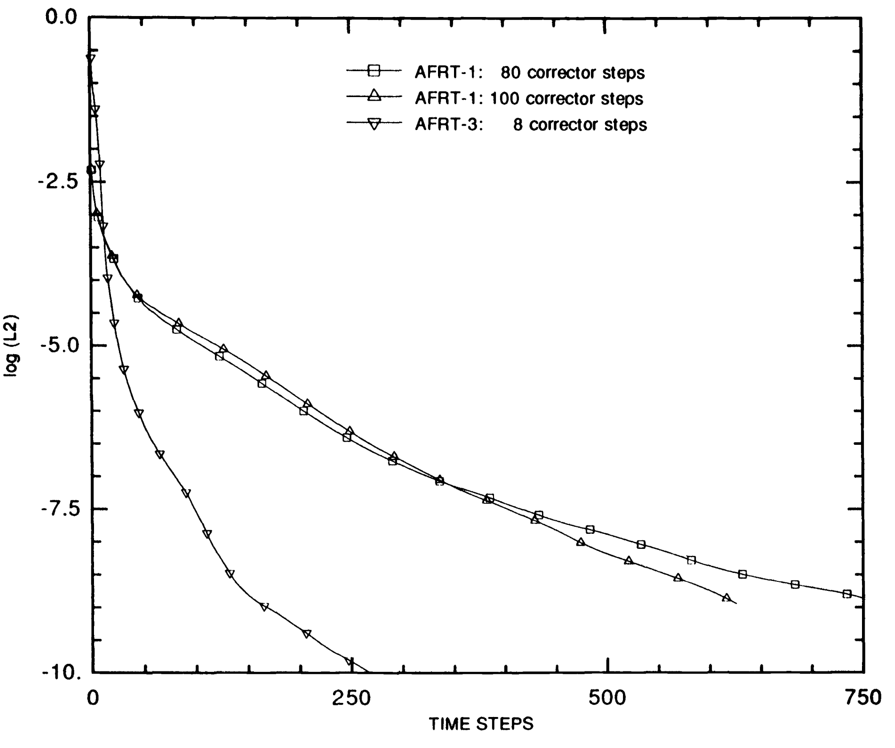


FIG. 10. *L_2 norm versus time steps for* AFRT-1 *and* AFRT-3 *with* $\Delta t = 0.06$ s.

solutions obtained by using AFRT-3 converged slightly faster than solutions obtained by using AFRT-1, provided that more than one corrector step is taken in AFRT-3.

Figures 5 and 6 show the results obtained by using $\Delta t = 0.01$ s for AFRT-1 and AFRT-3, respectively. The results shown in these figures are qualitatively similar to those shown in Figs. 3 and 4. As can be seen in Fig. 5, when AFRT-1 is used, one or two corrector steps improve the convergence rate, but more than two corrector steps bring no further improvements. This indicates that the iteration process used in AFRT-1 converges in two iterations when $\Delta t = 0.01$ s, compared to only one iteration when $\Delta t = 0.008$ s. Figure 6 shows that for AFRT-3 the use of only one corrector step has little effect on the convergence rate, but two and three corrector steps bring considerable improvements. If three corrector steps are used, the number of time steps needed to reach steady state is half of that needed by the original ADI three-factored scheme. This indicates that AF errors are quite significant for this larger time-step size. Note that the time-step size used here was the largest one for which the ADI three-factored scheme was able to yield converged solutions without the use of AF-error-reduction techniques. This time-step size can therefore be considered as the practical stability limit for the three-factored ADI scheme used in this study.

Figure 7 shows the results obtained by using $\Delta t = 0.015$ s. This figure shows the results for both AFRT-1 and AFRT-3. For this time-step size the three-factored ADI scheme was found to diverge. In Fig. 7 it can be seen that when AFRT-1 was used, a converged solution was obtained if one corrector step or four corrector steps were used. However, if two or three corrector steps were used, the solution diverged. For AFRT-3 it was found that two corrector steps were needed in order to obtain a converged solution and that the convergence rate improved as the number of corrector steps increased.

Figure 8 shows the results obtained by using $\Delta t = 0.02$ s. For this time-step size, when AFRT-1 was used, six corrector steps were needed to stabilize the ADI three-factored scheme. If AFRT-3 was used, then only two corrector steps were needed to stabilize the scheme. Again, when AFRT-3 was used, the convergence rate improved as the number of corrector steps was increased.

Figure 9 shows the results obtained by using $\Delta t = 0.03$ s. This time-step size is three times the practical stability limit for the ADI three-factored scheme. For this time-step size, when AFRT-1 was used, 10 corrector steps were needed to stabilize the scheme. Increasing the number of corrector steps to 30 improved the convergence rate significantly. When AFRT-3 was used, three corrector steps were needed to stabilize the scheme, and this resulted in a considerably better convergence rate than that obtained by using AFRT-1.

Figure 10 shows the results obtained by using $\Delta t = 0.06$ s, which is six times the practical stability limit for the ADI three-factored scheme. It was found that when AFRT-1 was used, at least 80 corrector steps were needed in order to stabilize the scheme. Increasing the number of corrector steps to 100 brought only a moderate improvement in convergence rate. When AFRT-3 was used, eight corrector steps were found to be adequate in stabilizing the scheme. Thus AFRT-3 has a considerably better convergence rate than does AFRT-1. However, it should be noted that the operation count needed to execute the corrector step in AFRT-3 increases quadratically with the number of corrector steps, whereas in AFRT-1 that increase is linear. This rapid increase in operation found in AFRT-3 is due to the iterations that must be performed within each corrector step.

The results of the numerical experiments described above indicate that the AF-error-reduction methods presented in this study are capable of reducing AF errors and increasing convergence rate. All results were obtained by applying the AF-error-reduction methods to a very stringent test problem, one in which the algorithm is unconditionally unstable. If the AF-error-reduction methods presented here were applied to algorithms that are inherently stable, better performance could be expected.

6. Concluding remarks. In this paper three methods were presented that can be used to reduce or eliminate AF errors that exist in two- and three-factored schemes such as the ADI, LU, and LU-SSOR schemes. These AF-error-reduction methods were illustrated and tested in the framework of the unconditionally unstable three-factored ADI scheme applied to the linear advection equation. Also, the convergence and stability properties of one AF-error-reduction method were analyzed.

The results of the tests and analyses indicate that the AF-error-reduction methods presented are capable of reducing AF errors and of increasing convergence rate. Thus the methods developed should be useful in stabilizing the commonly used unconditionally unstable three-factored ADI scheme, which is currently being stabilized by adding artificial dissipation. For factored schemes that are inherently stable, the AF-error-reduction methods should be useful for problems in which transient solutions are of interest and AF errors are larger than the time-discretization and time-linearization errors. When only steady-state solutions are of interest, the methods presented here can be used to accelerate convergence to steady state.

Finally, it is noted that the AF-error-reduction techniques presented can easily be incorporated into existing codes using two- and three-factored schemes. Also, there are several ways to improve the computational efficiency of these methods. For example, the number of iterations performed within each corrector step of AFRT-3 can be optimized. Also, convergence acceleration techniques, such as relaxation, can be incorporated into the iterative parts of AFRT-1 and AFRT-3.

Acknowledgments. The authors thank Richard J. Roelke, NASA Lewis Research Center, and they also thank an anonymous referee for careful reading of the manuscript and helpful suggestions.

REFERENCES

[1] R. M. Beam and R. F. Warming, *An implicit factored scheme for the compressible Navier–Stokes equations*, AIAA J., 16 (1978), pp. 393–402.

[2] W. R. Briley and H. McDonald, *On the structure and use of linearized block implicit schemes*, J. Comput. Phys., 34 (1980), pp. 54–73.

[3] T. H. Pulliam and D. S. Chaussee, *A diagonal form of an implicit approximate-factorization algorithm*, J. Comput. Phys., 39 (1981), pp. 347–363.

[4] D. A. Caughey, *Diagonal implicit multigrid algorithm for the Euler equations*, AIAA J., 26 (1988), pp. 841–851.

[5] J. L. Steger and R. F. Warming, *Flux Vector Splitting of the Inviscid Gasdynamics Equations with Application to Finite Difference Methods*, NASA Tech. Memo 78605; J. Comput. Phys., 40 (1981), pp. 263–293.

[6] A. Jameson and E. Turkel, *Implicit schemes and LU decompositions*, Math. Comp., 37 (1981), pp. 385–397.

[7] J. L. Thomas, B. van Leer, and R. W. Walter, *Implicit flux-split schemes for the Euler equations*, AIAA J., 10 (1985), pp. 85–1680.

[8] S. Yoon and A. Jameson, *Lower-upper symmetric-Gauss–Seidel method for the Euler and Navier–Stokes equations*, AAIA J., 26 (1988), pp. 1025–1026.

[9] S. X. Ying, *Three-dimensional implicit approximately factored schemes for the equations of gasdynamics*, Ph.D. thesis, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, 1986.

[10] D. L. Dwoyer and F. C. Thames, *Accuracy and stability of time-split finite-difference schemes*, AIAA, 1981, pp. 81–1005.

[11] R. F. Warming and R. M. Beam, *An extension of A-stability to alternating direction implicit methods*, BIT, 19 (1979), pp. 395–417.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1237–1252, September 1993

HIGH-ORDER SYMPLECTIC RUNGE–KUTTA–NYSTRÖM METHODS*

M. P. CALVO† AND J. M. SANZ-SERNA†

Abstract. A numerical method for ordinary differential equations is called symplectic if, when applied to Hamiltonian problems, it preserves the symplectic structure in phase space, thus reproducing the main qualitative property of solutions of Hamiltonian systems. The authors construct and test symplectic, explicit Runge–Kutta–Nyström (RKN) methods of order 8. The outcome of the investigation is that existing high-order, symplectic RKN formulae require so many evaluations per step that they are much less efficient than conventional eighth-order nonsymplectic, variable-step-size integrators even for low accuracy. However, symplectic integration is of use in the study of qualitative features of the systems being integrated.

Key words. Runge–Kutta–Nyström methods, symplectic integration, Hamiltonian problems, order conditions

AMS subject classifications. primary 65L05; secondary 70H05

1. Introduction.

In this paper we are concerned with Runge–Kutta–Nyström (RKN) methods for the numerical integration of second-order systems of differential equations of the special form

$$d^2\mathbf{y}/dt = \mathbf{f}(\mathbf{y}), \qquad y = [y^1, y^2, \dots, y^N]^T, \tag{1.1}$$

or, equivalently, of first-order systems

$$\frac{d\mathbf{y}}{dt} = \dot{\mathbf{y}}, \qquad \frac{d\dot{\mathbf{y}}}{dt} = \mathbf{f}(\mathbf{y}).$$

For the RKN formula specified by the tableau

$$\begin{array}{c|ccc} \gamma_1 & \alpha_{11} & \cdots & \alpha_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_s & \alpha_{s1} & \cdots & \alpha_{ss} \\ \hline & b_1 & \cdots & b_s \\ \hline & \beta_1 & \cdots & \beta_s \end{array}, \tag{1.2}$$

the equations that describe the step $t_n \to t_{n+1} = t_n + h$ take the form

$$\mathbf{Y}_i = \mathbf{y}_n + h\gamma_i\dot{\mathbf{y}}_n + h^2\sum_{j=1}^{s}\alpha_{ij}\mathbf{f}(\mathbf{Y}_j),$$

$$\dot{\mathbf{y}}_{n+1} = \dot{\mathbf{y}}_n + h\sum_{i=1}^{s} b_i\mathbf{f}(\mathbf{Y}_i),$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\dot{\mathbf{y}}_n + h^2\sum_{i=1}^{s}\beta_i\mathbf{f}(\mathbf{Y}_i),$$

* Received by the editors April 20, 1992; accepted for publication (in revised form) September 9, 1992. This work was supported by Junta de Castilla y León project 1031-89 and by Dirección General de Investigación Cientifica y Técnica project PB89-0351.

† Departamento de Matemática Aplicada y Computacíon, Facultad de Ciencias, Universidad de Valladolid, Valladolid, Spain.

where $\mathbf{Y}_i$ denote the internal stages. Throughout the paper we suppose that in (1.2)

$$\beta_i = b_i(1-\gamma_i), \qquad 1 \le i \le s; \tag{1.3}$$

this is a standard assumption that significantly decreases the number of order conditions that must be imposed on the method coefficients to ensure a given order of consistency; see [10, Chap. 2, Lemma 13.13].

If the function $\mathbf{f}$ in (1.1) is the gradient of a scalar potential $-V = -V(\mathbf{y})$ and we set $\mathbf{p} = \dot{\mathbf{y}}$, $\mathbf{q} = \mathbf{y}$, then (1.1) may obviously be rewritten as

$$\frac{dp^I}{dt} = -\frac{\partial V}{\partial q^I}, \qquad \frac{dq^I}{dt} = p^I, \quad 1 \le I \le N. \tag{1.4}$$

This is the Hamiltonian system of ordinary differential equations

$$\frac{dp^I}{dt} = -\frac{\partial H}{\partial q^I}, \qquad \frac{dq^I}{dt} = \frac{\partial H}{\partial p^I}, \quad 1 \le I \le N$$

with Hamiltonian function

$$H = H(\mathbf{p},\mathbf{q}) = T(\mathbf{p}) + V(\mathbf{q}), \quad T(\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{p}.$$

In mechanics the q variables represent Lagrangian coordinates, the p variables represent the corresponding momenta, T represents the potential energy, V represents the potential energy and H represents the total energy.

The recent literature has devoted much attention to the integration of Hamiltonian systems by means of canonical or symplectic methods; see [17] for a survey. A one-step numerical method is said to be canonical or symplectic if it preserves the so-called symplectic structure of the space of variables $(\mathbf{p}, \mathbf{q})$, thus reproducing the main qualitative property of solutions of Hamiltonian systems [2]. Suris [19] showed that the RKN method (1.2) is symplectic when applied to systems (1.4) if the coefficients satisfy the relations

$$b_i(\beta_j - \alpha_{ij}) = b_j(\beta_i - \alpha_{ji}), \qquad 1 \le i,j \le s; \tag{1.5}$$

see also [13]. On the other hand, if (1.2) does not possess redundant stages, (1.5) is also necessary for symplecticness; a rigorous proof of this necessity can be seen in [3] (cf. [1, §5]). In the remainder of the paper we use the expression "symplectic RKN method" to refer to RKN methods (1.2) that satisfy (1.5).

Okunbor and Skeel [14] studied the families of explicit, symplectic RKN methods with one, two, or three stages. The present authors [4], [5], [7] have constructed and tested an explicit, five-stage, fourth-order symplectic RKN method with optimized error constants. This method uses four function evaluations per step: the evaluation for the fifth stage of the current step coincides with the first evaluation in the next step (FSAL (first same as last) technique). In [15] Okunbor and Skeel construct explicit symplectic RKN formulae with five stages and seven stages and orders 5 and 6, respectively. For separable Hamiltonian systems [1], Yoshida [20] derives explicit, symplectic methods with order 8. When applied to problems of the form (1.1), Yoshida's methods reduce to RKN schemes of order 8 with 16 stages and 15 evaluations per step.

The experiments in [7] show that in the accurate long-time integration of problems of the form (1.4) the constant-step-size implementation of the fourth-order symplectic formula constructed there is more efficient than a variable-step-size, fourth-order code based on an embedded RKN pair due to Dormand, El-Mikkawy, and Prince [8], [9]. The purpose of the present paper is to construct explicit, symplectic RKN methods of order 8 and to compare them with standard nonsymplectic RKN codes of the same order. The outcome of our investigation is that existing high-order, symplectic RKN formulae require so many evaluations per step that they are much less efficient than conventional eighth-order nonsymplectic, variable-step-size integrators, even for low accuracy. However, symplectic integration is of use in the study of qualitative features of the systems being integrated.

The structure of the paper is as follows. Section 2 reviews the theory of order conditions for symplectic RKN. Section 3 deals with the simplifying assumptions used later in the derivation of methods. In §4 we present a family of explicit, symplectic RKN methods. Specific order-7 methods within this family are constructed in §5. Section 6 is devoted to order-8 formulae, and §7 contains some numerical illustrations.

2. Order conditions for symplectic RKN methods. The conditions that must be imposed for an RKN method (1.2) for (1.1) to have order $\geq r$ are well known; the reader is referred to [10], whose terminology we follow. There is an order equation for each (rooted) SN-tree with r or fewer vertices (recall that (1.3) is assumed throughout). As an illustration, we have depicted in Fig. 1 the 10 SN-trees with six vertices. Furthermore, Table 1 displays the number m of SN-trees with r vertices, $1 \leq r \leq 10$. Clearly, for (1.2) to have order $\geq r$, the required number of order conditions is $\sum_{i=1}^{r} m_i$, a quantity that has also been tabulated in Table 1. Apparently, the generating function $M(z) = \sum_{i=1}^{\infty} m_i z^i$ for the sequence $\{m_r\}_{r=1}^{\infty}$ was first studied in [6].

In [6] we proved that the symplecticness conditions (1.5) act as simplifying assumptions, i.e., when (1.5) holds, not all order conditions are independent and some of them are implied by the remaining ones. For instance, for a symplectic method with order ≥ 5, the order condition associated with the tree $t_{6,2}$ is equivalent to the order condition associated with $t_{6,8}$. This comes about because $t_{6,2}$ and $t_{6,8}$ consist of the same vertices and edges and differ only in the location of the root. In other words, $t_{6,2}$ and $t_{6,8}$ are the same as unrooted SN-trees. For the same reason there is equivalence between $t_{6,3}$ and $t_{6,9}$, between $t_{6,4}$ and $t_{6,5}$, and among $t_{6,6}$, $t_{6,7}$ and $t_{6,10}$. Thus for a symplectic method with order ≥ 5 to have order 6 it is enough to impose five order conditions, one for each equivalence class $\{t_{6,1}\}$, $\{t_{6,2}, t_{6,8}\}$, $\{t_{6,3}, t_{6,9}\}$, $\{t_{6,4}, t_{6,5}\}$, $\{t_{6,6}, t_{6,7}, t_{6,10}\}$. The number m_r^* of corresponding equivalence classes for SN-trees with r vertices, $1 \leq r \leq 10$, is given in Table 1. The accumulated quantity $\sum_{i=1}^{r} m_i^*$ gives the total number of conditions for (1.2) subject to (1.5) to have order $\geq r$. A comparison of the third and fifth columns of Table 1 bears out the substantial reduction in order conditions implied by symplecticness.

3. Standard simplifying assumptions. Let us now leave aside the symplecticness conditions (1.5) and consider the well-known simplifying assumptions [10, Chap. 2, Lemma 13.14]

$$\sum_{j=1}^{s} \alpha_{ij} = \frac{\gamma_i^2}{2}, \qquad 1 \leq i \leq s, \tag{3.1}$$

that are often used in the construction of high-order RKN methods. When (1.2) satisfies (3.1), it is possible to disregard the order conditions associated with SN-trees with two or more vertices where at least one end vertex is fat. The order conditions for such trees are equivalent to order conditions for trees where all end vertices are meager. The basis

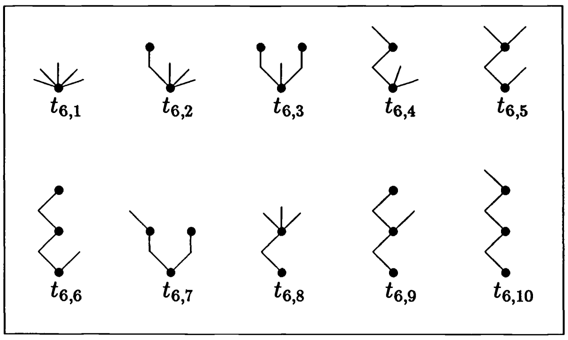


FIG. 1. SN-*trees with six vertices.*

TABLE 1

r	m_r	$\sum_{i=1}^{r} m_i$	m_r^*	$\sum_{i=1}^{r} m_i^*$	m_i'	$\sum_{i=1}^{r} m_i'$	$m_i'^*$	$\sum_{i=1}^{r} m_i'^*$
1	1	1	1	1	1	1	1	1
2	1	2	1	2	1	2	1	2
3	2	4	2	4	1	3	1	3
4	3	7	2	6	2	5	1	4
5	6	13	4	10	3	8	2	6
6	10	23	5	15	5	13	2	8
7	20	43	10	25	9	22	4	12
8	36	79	14	39	15	37	5	17
9	72	151	27	66	27	64	9	26
10	137	288	43	109	48	112	13	39

for this equivalence is illustrated in Fig. 2, where the order conditions for both trees are equivalent provided that the circle with the three branches at the bottom denotes in both cases the same arbitrary SN-tree. By iteration of the reduction in Fig. 2, the order condition for any tree with two or more vertices can be seen to be equivalent to order conditions for trees with only meager end vertices. For instance, in Fig. 1 the order condition for $t_{6,2}$ is the same as the order condition for $t_{6,1}$ and may be disregarded. For analogous reasons the order conditions for $t_{6,3}$, $t_{6,6}$, $t_{6,7}$, $t_{6,9}$ may be ignored, and this leaves five trees with six vertices to be considered. For general r we have the following result (caution: a prime does not mean differentiation!).

THEOREM 3.1. *Let m_r', $r \geq 2$ denote the number of* SN-*trees with r vertices without fat end vertices, and set $m_1' = 1$. Then*

$$m_r' = \sum_{\substack{k, j_1, j_3, \ldots, j_k \\ j_1 + 3j_3 + \cdots + kj_k = r-1}} \binom{m_2' + j_3 - 1}{j_3} \cdots \binom{m_{k-1}' + j_k - 1}{j_k}, \tag{3.2}$$

and the corresponding generating function $M'(z) = \sum_{r=1}^{\infty} m_r' z^r$ satisfies the equation

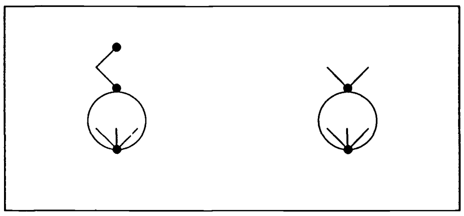


FIG. 2. *Equivalent* SN-*trees if* (3.1) *holds.*

$$M'(z) = \frac{z}{(1-z)(1-z^3)^{m'_2}\cdots(1-z^k)^{m'_{k-1}}\cdots}. \tag{3.3}$$

Proof. For $r \geq 2$ consider a special Nyström tree t with r vertices and remove its root. This gives rise to, say, j_1 graphs with one vertex, j_2 graphs with two vertices, etc. If t had no fat end vertex, then $j_2 = 0$ and for $k \geq 2$ each among the j_k graphs with k vertices consist of a meager vertex (that was a child of the root in the original t) followed by a special Nyström tree of order $k-1$ where all end vertices are meager. Hence for $k \geq 2$ the j_k graphs with k vertices can be chosen in

$$\binom{m'_{k-1} + j_k - 1}{jk}$$

different ways. This leads to (3.2). Equation (3.3) is a direct consequence of (3.2), along with the formula

$$\frac{1}{(1-z^r)^m} = \sum_{j=0}^{\infty} \binom{m+j-1}{j} z^{jr}. \qquad \square$$

The theorem makes it possible to recursively compute the m'_r. These have been tabulated in Table 1 for $1 \leq r \leq 10$. A comparison of the values of the quantities $\sum m_i^*$ and $\sum m'_i$ reveals that (1.5) and (3.1) leave roughly the same number of independent conditions to be considered. Therefore, in a sense (1.5) and (3.1) are as effective as simplifying assumptions. However, in (3.1) there are only s conditions to be imposed, whereas (1.5) comprises $s(s-1)/2$ relations (note that i and j play a symmetric role). Thus if one is not interested in Hamiltonian problems, (3.1) should clearly be preferred to (1.5). On the other hand, if to achieve symplecticness we impose (1.5), then we have reduced the number of order conditions by roughly the same amount we would have reduced that number by imposing the familiar simplifying assumptions (3.1).

The question arises of what happens when *both* (1.5) and (3.1) hold. For instance, for $r = 6, t_{6,1}, t_{6,2}, t_{6,3}$ are equivalent after (3.1) and $\{t_{6,2}, t_{6,8}\}$ and $\{t_{6,3}, t_{6,9}\}$ are equivalence classes for (1.5), so that $t_{6,1}, t_{6,2}, t_{6,3}, t_{6,8}, t_{6,9}$ all become equivalent. For the same reason, the order conditions for the remaining order-6 trees $t_{6,4}, t_{6,5}, t_{6,6}, t_{6,7}, t_{6,10}$ form a second equivalence class. Hence under (1.5) and (3.1) there are only two order conditions arising from order-6 trees. For general r let us say that two SN-trees t, t^* r vertices are S-equivalent if there exist a sequence of SN-trees $t_1, t_2, \ldots, t_k$ with $t_1 = t, t_k = t^*$, where t_i and t_{i+1}, $1 \leq i \leq k-1$, either are related as in Fig. 2 or differ only in the location of the root. Thus if (1.5) and (3.1) hold and (1.2) has order $\geq r-1$, then order conditions for t and t^* are equivalent whenever t and t^* are S-equivalent.

THEOREM 3.2. *Let $m_r'^*$ denote the number of equivalence classes of* SN-*trees of order r under the relation* S. *Then the generating function $M'^*(z) = \sum_{r=1}^{\infty} m_r'^* z^r$ is given by*

$$M'^*(z) = M'(z) - \tfrac{1}{2}z(M'(z)^2 - M'(z^2)). \tag{3.4}$$

Proof. For a given $r \geq 2$ let us consider the m_r^* free or unrooted SN-trees. There is one such tree for each equivalence class based on (1.5). Those free trees that have one or more fat end vertex can be deleted in view of (3.1). Our task is to count the free trees that remain after such a deletion. As in [6], we resort to the notion of centroid of a free tree; see, e.g., [12], [18].

The following cases are possible for the free trees that remain.

(i) There is one centroid that is a meager vertex. By chopping off the centroid we obtain two (rooted) SN-trees. These must have the same order j, in view of the definition of centroid. Hence in this case r must be odd and $j = (r-1)/2$. For r odd $r > 3$ in view of the definition of m_j', $j \geq 2$, there are

$$\binom{m_{r-1/2}' + 1}{2} = \frac{1}{2} m_{(r-1)/2}'(m_{(r-1)/2}' + 1) \tag{3.5}$$

free trees in this case. For $r = 3$ obviously there is no free tree in this case.

(ii) There are two centroids. In this situation r must be even and the centroids are adjacent; one of them is fat and the other is meager. Chop off the meager centroid to get a rooted SN-tree of order $r/2$ without fat end vertices and to get a rooted SN-tree of order $r/2-1$ without fat end vertices. Therefore, for r even, $r > 4$, there are

$$m_{r/2}' m_{r/2-1}' \tag{3.6}$$

free trees in this category. For $r = 4$ this category is empty.

(iii) There is one centroid, and this is fat. The number of equivalence classes in this case is

$$m_r' - (m_1' m_{r-2}' + \cdots + m_{r-\max(3,r/2)}' m_{\max(3,r/2)}') \tag{3.7}$$

if r is even and

$$m_r' - (m_1' m_{r-2}' + \cdots + m_{r-\max(3,(r+1)/2)}' m_{\max(3,(r+1)/2)-1}') \tag{3.8}$$

if r is odd. These formulae are proved by an argument similar to that used to obtain [12, §2.3.4.4, formula (8)].

Formula (3.4) is a consequence of (3.5)–(3.8). $\square$

The last column in Table 1 bears out the important reduction in the number of order conditions brought about by the combination of (1.5) and (3.1). In Fig. 3 we have depicted representatives of the 12 S-classes of equivalence to be considered for order ≥ 7, together with the corresponding order conditions.

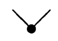
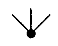
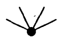
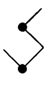
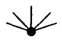
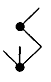
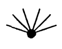
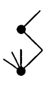
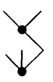
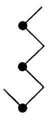
	$\tau_{1,1}$	$\Phi_{1,1} \equiv \Sigma b_i - 1 = 0$
	$\tau_{2,1}$	$\Phi_{2,1} \equiv \Sigma b_i \gamma_i - 1/2 = 0$
	$\tau_{3,1}$	$\Phi_{3,1} \equiv \Sigma b_i \gamma_i^2 - 1/3 = 0$
	$\tau_{4,1}$	$\Phi_{4,1} \equiv \Sigma b_i \gamma_i^3 - 1/4 = 0$
	$\tau_{5,1}$	$\Phi_{5,1} \equiv \Sigma b_i \gamma_i^4 - 1/5 = 0$
	$\tau_{5,2}$	$\Phi_{5,2} \equiv \Sigma b_i \gamma_i \alpha_{ij} \gamma_j - 1/30 = 0$
	$\tau_{6,1}$	$\Phi_{6,1} \equiv \Sigma b_i \gamma_i^5 - 1/6 = 0$
	$\tau_{6,2}$	$\Phi_{6,2} \equiv \Sigma b_i \gamma_i^2 \alpha_{ij} \gamma_j - 1/36 = 0$
	$\tau_{7,1}$	$\Phi_{7,1} \equiv \Sigma b_i \gamma_i^6 - 1/7 = 0$
	$\tau_{7,2}$	$\Phi_{7,2} \equiv \Sigma b_i \gamma_i^3 \alpha_{ij} \gamma_j - 1/42 = 0$
	$\tau_{7,3}$	$\Phi_{7,3} \equiv \Sigma b_i \gamma_i^2 \alpha_{ij} \gamma_j^2 - 1/84 = 0$
	$\tau_{7,4}$	$\Phi_{7,4} \equiv \Sigma b_i \gamma_i \alpha_{ij} \alpha_{jk} \gamma_k - 1/840 = 0$

FIG. 3. *Order conditions for order* ≥ 7 *under* (1.5) *and* (3.1).

4. Family of explicit, symplectic RKN methods. Explicit symplectic RKN methods are of the form [14]

$$
\text{(4.1)} \qquad
\begin{array}{c|cccccc}
\gamma_1 & 0 & 0 & \cdots & 0 & 0 \\
\gamma_2 & b_1(\gamma_2-\gamma_1) & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \iddots & \vdots & \vdots \\
\gamma_{s-1} & b_1(\gamma_{s-1}-\gamma_1) & b_2(\gamma_{s-1}-\gamma_2) & \cdots & 0 & 0 \\
\gamma_s & b_1(\gamma_s-\gamma_1) & b_2(\gamma_s-\gamma_2) & \cdots & b_{s-1}(\gamma_s-\gamma_{s-1}) & 0 \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s \\
\hline
 & \beta_1 & \beta_2 & \cdots & \beta_{s-1} & \beta_s
\end{array}
$$

subject to (1.3). Thus with s stages there are $2s$ free parameters. For $s \geq 2$ we consider the subclass of methods given by

$$\gamma_1 = 0, \qquad \gamma_s = 1 \tag{4.2}$$

and

$$b_1 = \gamma_2/2; \quad b_i = (\gamma_{i+1} - \gamma_{i-1})/2, \quad 2 \leq i \leq s-1; \quad b_s = (1 - \gamma_{s-1})/2. \tag{4.3}$$

Note that (4.2) and (4.3) leave only $s - 2$ free parameters $\gamma_2, \dots, \gamma_{s-1}$ in (4.1). For arbitrarily fixed values of $\gamma_2, \dots, \gamma_{s-1}$, the following properties are easily verified.

(i) Method (4.1)–(4.3) has the FSAL property: the sth stage of the current step coincides with the first stage of the next step. Thus (4.1)–(4.3) effectively require $s - 1$ evaluations per step.

(ii) Method (4.1)–(4.3) satisfies the standard simplifying assumptions (3.1).

(iii) Method (4.1)–(4.3) has order ≥ 2, i.e., the conditions $\sum b_i = 1$, $\sum b_i \gamma_i = \frac{1}{2}$ are implied by the structure of tableau (4.1) and relations (4.2) and (4.3).

It is also useful to observe that a step of length h with (4.1)–(4.3) is a concatenation of $s - 1$ steps of successive lengths $(\gamma_2 - \gamma_1)h, (\gamma_3 - \gamma_2)h, \dots, (\gamma_s - \gamma_{s-1})h$ with the simple method

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & 0 \end{array} \tag{4.4}$$

that results after setting $s = 2$ in (4.1)–(4.3). Let us be more precise. Let ψ_h represent the transformation in $(\dot{\mathbf{y}}, \mathbf{y})$-space that effects a step of length h with (4.1)–(4.3), i.e., $(\dot{\mathbf{y}}_{n+1}, \mathbf{y}_{n+1}) = \psi_h(\dot{\mathbf{y}}_n, \mathbf{y}_n)$ if $(\dot{\mathbf{y}}_{n+1}, \mathbf{y}_{n+1})$ is the result of a step of length h from the preceding approximation $(\dot{\mathbf{y}}_n, \mathbf{y}_n)$. Let $\psi_h^{[2]}$ represent the corresponding transformation for (4.4). Then

$$\psi_h = \psi^{[2]}_{(\gamma_s - \gamma_{s-1})h} \cdots \psi^{[2]}_{(\gamma_2 - \gamma_1)h}. \tag{4.5}$$

This formula makes it easy to find the adjoint method ψ_h^* of ψ_h. Recall that by definition [10] ψ_h^* is the method such that ψ^*_{-h} inverts ψ_h, i.e., a step of length h with (4.1)–(4.3) followed by a step of length $-h$ with the adjoint method of (4.1)–(4.3) leaves the numerical solution unchanged. From (4.5)

$$\psi_h^* = [\psi^{[2]}_{(\gamma_s - \gamma_{s-1})h} \cdots \psi^{[2]}_{(\gamma_2 - \gamma_1)h}]^* = \psi^{[2]*}_{(\gamma_2 - \gamma_1)h} \cdots \psi^{[2]*}_{(\gamma_s - \gamma_{s-1})h},$$

but (4.4) is easily seen to be selfadjoint and hence

$$\psi_h^* = \psi^{[2]}_{(\gamma_2 - \gamma_1)h} \cdots \psi^{[2]}_{(\gamma_s - \gamma_{s-1})h}.$$

Comparison with (4.5) reveals that ψ_h^* is the method of the family (4.1)–(4.3) based on the abscissae $\gamma_2^*, \dots, \gamma_{s-1}^*$ defined by $\gamma_i^* = 1 - \gamma_{s+1-i}$.

5. Constructing seventh-order methods. In this section we describe our experience in constructing order-7 methods of the form (4.1)–(4.3). Since both (1.5) and (3.1) hold, the last column in Table 1 shows that there are 12 order conditions to be imposed (see Fig. 3). However, the order conditions $\Phi_{1,1} = 0$, $\Phi_{2,1} = 0$ that guarantee order ≥ 2 are automatically satisfied, so that the choice of the $s - 2$ free parameters $\gamma_2, \dots, \gamma_{s-1}$ should be directed toward enforcing the 10 remaining conditions

$$\Phi_{3,1} = 0, \quad \Phi_{4,1} = 0, \dots, \quad \Phi_{7,4} = 0. \tag{5.1}$$

This suggests $s \geq 12$. The choice $s = 12$ leaves no freedom to tune the formula, and we set $s = 13$. We therefore undertook the task of numerically solving the nonlinear system (5.1) comprising 10 equations in the 11 unknowns $\gamma_2, \dots, \gamma_{12}$. The solutions form curves in $\mathcal{R}^{11}$ that can be followed by continuation once a particular solution has been found.

Finding initial solutions of (5.1) to start the continuation procedure was not an easy task. After many unsuccessful attempts the following strategy was adopted. We began by considering the function

$$\lambda = \Phi_{3,1}^2 + \Phi_{4,1}^2 + \Phi_{5,1}^2 + \Phi_{5,2}^2 + \Phi_{6,1}^2$$

(see Fig. 3) of the variables $\gamma_2, \dots, \gamma_{12}$. We minimized λ subject to bounds $-5 \leq \gamma_i \leq 5$, $2 \leq i \leq 12$, and to the equality constraints $\Phi_{6,2} = \Phi_{7,1} = \Phi_{7,2} = \Phi_{7,3} = \Phi_{7,4} = 0$. To this end the NAG routine E04UCF was used with the starting values for $\gamma_2, \dots, \gamma_{12}$ generated randomly with the NAG routine G05DAF. Clearly, a solution of (5.1) is found whenever the objective function γ is successfully brought to its global minimum $\lambda = 0$ by the minimization routine.

For the continuation procedure we used one of the unknowns $\gamma_2, \dots, \gamma_{12}$ as a continuation parameter. The particular unknown to be used at each step of the continuation procedure was determined as follows. Gaussian elimination with column pivoting was performed in the 10×11 Jacobian matrix of the system (5.1) evaluated at the current value of the solution. The parameter was chosen to be the unknown that was not used as a pivot, i.e., the unknown whose column would be in the 11th place if the columns were actually interchanged to carry out the pivoting. In a sense this identifies the unknown that is (locally in the solution curve) least constrained by (5.1) and that therefore is (locally) best suited for parametrizing the solutions of the system. Once the index $i_0, 2 \leq i_0 \leq 12$, of the unknown to be used as a parameter has been determined, we solved by Newton's method the 11×11 system given by (5.1), along with the equation $\gamma_{i_0} = \gamma_{i_0}^0 + \delta$, where $\gamma_{i_0}^0$ is the value of γ_{i_0} at the current solution and $\delta = 0.01$ denotes the increment in the parameter.

The coefficients of a specific method constructed by following this methodology are presented later in the paper.

6. Eighth-order methods. Once a method ψ_h of the class (4.1)–(4.3) with order $r = 7$ and $s = 13$ has been obtained, it is possible to use it so as to have eighth-order symplectic integration. In fact, it is enough to consider the method [16]

$$\bar{\psi}_h = \psi_{h/2}^* \psi_{h/2}. \tag{6.1}$$

A step of length h with the new method $\bar{\psi}_h$ consists of a step of length $h/2$ with the given order-7 formula followed by a step of length $h/2$ with the adjoint formula. The method $\bar{\psi}_h$ is symplectic as obtained by concatenating symplectic formulae and obviously has order ≥ 7. Furthermore, $\bar{\psi}_h$ is clearly selfadjoint, so that it has even order. Hence $\bar{\psi}_h$

is an eighth-order method. Note that $\bar{\psi}_h$ uses 24 evaluations per step as ψ_h and ψ_h^* are FSAL methods with 13 stages. The ratio number of stages per order is 26/8 versus the minimum 17/8 suggested by Table 1, but the situation is not bad at all. The approximation obtained after taking the first half-step $\psi_{h/2}$ is also globally accurate of the eighth order; the first half-step starts from an approximation with (global) error $O(h^8)$ and introduces a local error $O(h^8)$, so that the global error after the first half-step is $O(h^8)$. Hence output with global accuracy $O(h^8)$ is available after every 12 function evaluations. On the other hand, having 24 evaluations per step is certainly a drawback if variable step sizes are used: if a step is rejected, too many evaluations are wasted. Fortunately, symplectic integration should be used with constant step sizes [4], [5], [7] and so we feel that to find eighth-order formulae it is better to resort to the technique in (6.1) than to look directly at methods of the family (4.1)–(4.3).

The parameter in the continuation procedure used to find the seventh-order method ψ_h is chosen for the method $\bar{\psi}_h$ in (6.1) to have small error constants (see [8]). The $\dot{\mathbf{y}}$-truncation error and $\mathbf{y}$-truncation error of an eighth-order RKN method such as $\bar{\psi}_h$ have, respectively, the forms

$$h^9 \sum_j \Phi'_{9,j} \mathbf{F}_{9,j} + O(h^{10}) \tag{6.2}$$

and

$$h^9 \sum_k \Phi_{9,k} \mathbf{F}_{8,k} + O(h^{10}), \tag{6.3}$$

where $\mathbf{F}_{8,k}$ and $\mathbf{F}_{9,j}$ are elementary differentials that depend only on the system (1.1) being integrated and $\Phi'_{9,j}$, $\Phi_{9,k}$ are polynomials in the method coefficients α_{ij}, γ_i, β_i, b_i. In (6.2) the sum is extended to the 72 SN-trees of order 9, and in (6.3) the sum is extended to the 36 SN-trees of order 8 (see Table 1). (Note that in (6.2) and (6.3) the coefficients that are featured are those of the order-8 method, whereas in (5.1) we deal with the coefficients of the order-7 method. Also, in (6.2) and (6.3) all SN-trees are considered, whereas in (5.1) we took only one SN-tree per S-class of equivalence.) We try to minimize the Euclidean norm N of the vector with $72+36$ components $\Phi'_{9,j}$, $\Phi_{9,k}$. To this end, at each step of the continuation procedure described in §5 we evaluate N for the eighth-order method obtained $\bar{\psi}_h$, by means of (6.1), from the current seventh-order method ψ_h. The following coefficients identify the formula ψ_h that, among those we found, leads to the $\bar{\psi}_h$ with the lowest $N (N = 1.6 \times 10^{-5})$:

$$\gamma_2 = 0.60715821186110352503,$$

$$\gamma_3 = 0.96907291059136392378,$$

$$\gamma_4 = -0.10958316365513620399,$$

$$\gamma_5 = 0.05604981994113413605,$$

$$\gamma_6 = 1.30886529918631234010,$$

$$\gamma_7 = -0.11642101198009154794,$$

$$\gamma_8 = -0.29931245499473964831,$$

$$\gamma_9 = -0.16586962790248628655, \tag{6.4}$$

$$\gamma_{10} = 1.22007054181677755238,$$

$$\gamma_{11} = 0.20549254689579093228,$$

$$\gamma_{12} = 0.86890893813102759275.$$

7. Numerical results. Although the numerical experiments presented in this section provide information on the advantages and disadvantages of symplectic integrators, they are limited in scope. More extensive testing is required before definite conclusions can be put forward.

We consider three explicit, symplectic methods, used with constant step sizes:

(i) S8: order 8, 26 stages, 24 evaluations per step, symplectic RKN formula associated with (6.4) by means as of (6.1). This has error constant $N = 1.6 \times 10^{-5}$.

(ii) Y8: order 8, 15 evaluations per step, symplectic method given in [20, Table 2, column D]. This is the order-8 method with the lowest value of N among those constructed in [20] and has $N = 4.4 \times 10^{-3}$.

(iii) S4: order 4, five stages, four evaluations per step, symplectic RKN formula constructed in [7].

As a reference standard (i.e., nonsymplectic) method we consider the following:

(iv) D8: order 8, nine stages, eight evaluations per step, RKN formula by Dormand, El-Mikkawy, and Prince [9, Table 1]. This has $N = 8.3 \times 10^{-7}$. An embedded order-6 method presented in [9] was used to estimate the error in a variable-step implementation. There are some printing errors in the method coefficients in the original [9], and the reader should see the corresponding corrigendum.

The values of N given above cannot be directly compared because the work per step is different for different methods. More informative error coefficient values can be obtained by assuming that a method with q function evaluations per step uses a step size of qh. If the method is of order p, this multiplies the values of N by q^p. For S8, Y8, and D8 the normalized values of the error coefficient turn out to be 1.8×10^6, 1.1×10^7, and 14, respectively. On raising these values to the power $-1/p = -1/8$, we obtain a crude measure of the efficiency of the various methods. The result is 0.16 for S8, 0.13 for Y8, and 0.71 for D8. The formulae S8 and Y8 are very demanding in function evaluations and are hence inefficient when compared with the nonsymplectic formula D8. This inefficiency is due to the $O(s^2)$ number of degrees of freedom in the RKN tableau that are used to enforce the symplecticness conditions (1.5).

Our first test problem was used in [7]. It corresponds to the Newton potential $V(q^1, q^2) = -1/\|\mathbf{q}\|$ (Kepler's problem) with initial condition

$$p^1 = 0, \quad p^2 - \sqrt{\frac{1+e}{1-e}}, \quad q^1 = 1 - e, \quad q^2 = 0.$$

Here e is the eccentricity of the orbit, $0 \leq e < 1$, that in the experiments to be reported is chosen to be $e = 0.5$. (The value of e does not significantly influence the outcome of the experiments.) The solution is 2π-periodic. Errors are measured in the Euclidean norm of $\mathcal{R}^4$.

Figure 4 corresponds to a final integration time $T = 810 \times 2\pi$ and depicts error at $t = T$ against number of function evaluations. The following runs are presented:

(i) S8, with step sizes $h = 2\pi/32, 2\pi/64, 2\pi/128$ (asterisks joined by a solid line).

(ii) Y8, with step sizes $h = 2\pi/64, 2\pi/128, 2\pi/256$ (plus signs joined by a dotted line).

(iii) S4, with step sizes $h = 2\pi/128, 2\pi/256, 2\pi/512, 2\pi/1024, 2\pi/2048$ ($\times$ joined by a dashed line).

(iv) D8, with absolute error tolerances $10^{-7}, 10^{-8}, 10^{-9}, 10^{-10}, 10^{-11}, 10^{-12}$ (circles joined by dash-dot line).

Clearly, S8 is more efficient than Y8. Yoshida's method Y8 is more efficient than the lower-order method S4 when small errors are required. Otherwise, S4 is more efficient than Y8. However, the nonsymplectic method S8 is clearly more efficient than any of the symplectic formulae tested.

Figure 5 is similar to Fig. 4, but now the final time $T = 21870 \times 2\pi$ is longer. The runs depicted are as follows:

(i) S8, with step sizes $h = 2\pi/32, 2\pi/64, 2\pi/128$.

(ii) Y8, with step sizes $h = 2\pi/128, 2\pi/256$.

(iii) S4, with step sizes $h = 2\pi/256, 2\pi/512, 2\pi/1024, 2\pi/2048$.

(iv) D8, with absolute error tolerances $10^{-9}, 10^{-10}, 10^{-11}, 10^{-12}$.

The conclusions as to the relative efficiency of the methods are the same as above. However, the advantage of D8 is not so marked as before. This is due to the better error propagation properties of symplectic integrators [7]. In Fig. 6 we have depicted error against t (measured in periods) for S8 ($h = 2\pi/32$), Y8 ($h = 2\pi/128$), S4 ($h = 2\pi/256$), and D8 (TOL $= 10^{-9}$). Note that in this figure different methods are working differently; only the slopes in the different lines should be compared. In the symplectic methods the error grows linearly with t, whereas in the nonsymplectic method D8 the growth is as t^2, as is proved rigorously in [7]. Therefore, as the final integration time T increases, S8 and Y8 improve their efficiency relative to D8. However, the crossover point T for which S8 becomes more efficient than D8 is too large: perhaps T corresponds to millions of periods of the planet whose motion is being integrated.

From this experiment we conclude that if accurate solutions are needed, even for long integration times, a high-order standard code may easily be a better choice than a symplectic algorithm. For high-order RKN integrators too many ($O(s^2)$) degrees of freedom in the tableau are sacrificed to achieve symplecticness, and this sacrifice makes the formula very expensive relative to standard RKN methods. This should be compared with the conclusions in [7], where it is shown that for Kepler's problem fourth-order symplectic integrators are more efficient than fourth-order variable-step standard codes. In [7] the work per step of the fourth-order symplectic algorithm is $\frac{4}{3}$ of the work per step of the reference standard fourth-order algorithm and the advantages of symplecticness

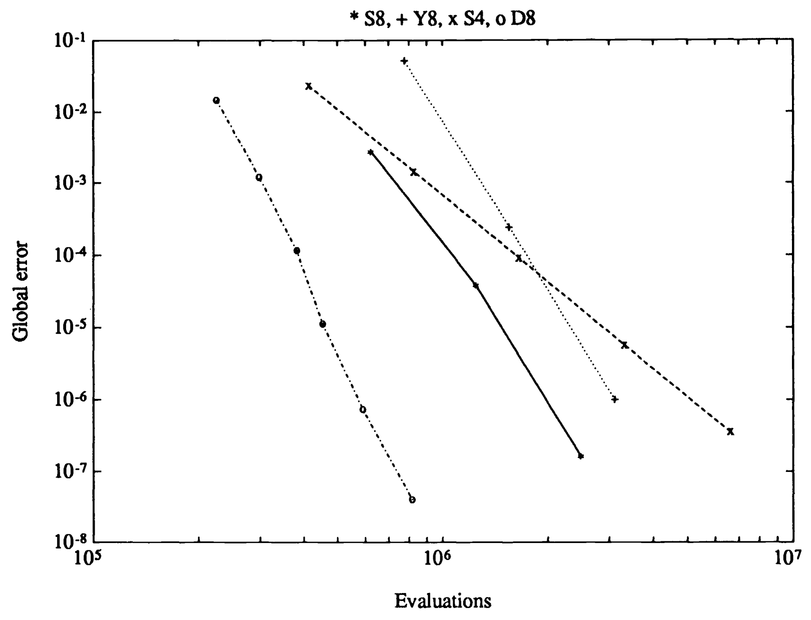


FIG. 4. *Efficiency plot at* $T = 810 \times 2\pi$.

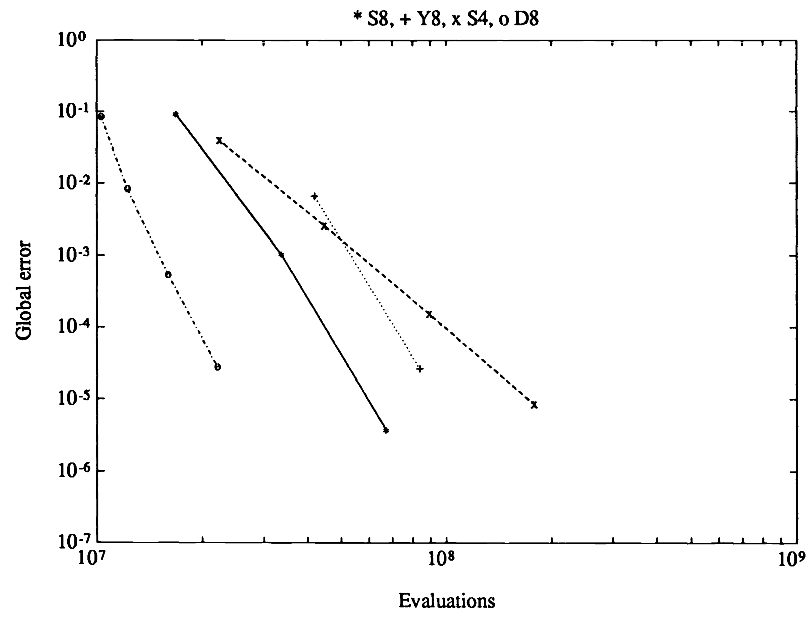


FIG. 5. *Efficiency plot at* $T = 21870 \times 2\pi$.

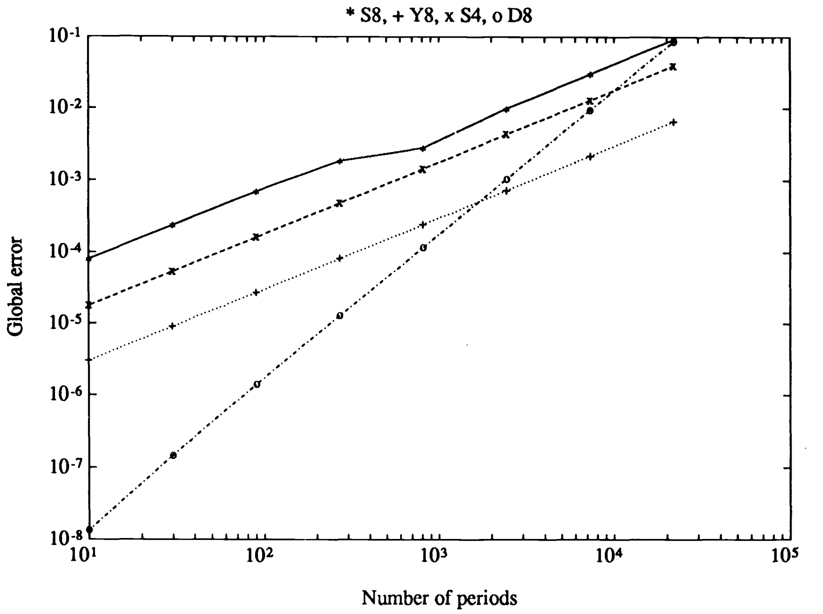


FIG. 6. *Error against t.*

make up for the increased cost per step. Here the work per step in S8 is three times the work per step in D8.

The second test problem is taken from Herbst and Ablowitz [11]. It originates from the sine-Gordon equation

$$u_{tt} - u_{xx} + \sin u = 0, \qquad 0 < x < L = 2\sqrt{2}\pi, \quad t > 0. \tag{7.1}$$

subject to periodic boundary conditions and to the initial conditions

$$u(x,0) = \pi + 0.1\cos(2\pi x/L), \qquad u_t(x,0) = 0. \tag{7.2}$$

Equation (7.1) may be thought of as describing the motion of a family of pendula. At each value of $x, 0 < x < L$, we have one pendulum. The term u_{xx} provides coupling between the motions of neighboring pendula. It represents a force that tries to keep a common value of the angle u for all the pendula. From the initial condition (7.2) we see that all pendula are initially left near the unstable equilibrium $u = \pi$. The pendula in $0 < x < L/4$ or $3L/4 < x < L$ start *above* the value $u = \pi$ and hence will increase u in order to approach the stable equilibrium at $u = 2\pi$. The pendula in $L/4 < x < 3L/4$ start *below* the value $u = \pi$ and will decrease u to approach the stable equilibrium at $u = 0$. This causes the term u_{xx} to become important. The effect of the restoring force is that the pendula are prevented from reaching the lowest $u = 2\pi$ or $u = 0$ positions and, rather, start going upward back to the initial positions, leading to a periodic motion. The solid curve in Fig. 7 represents u as a function of $t, 0 < t < 16L$, for the pendulum at $x = L/2$.

As in [11], (7.1) is discretized in space by the standard pseudospectral technique, with a mesh length $\Delta x = L/32$. This leads to a Hamiltonian system of the form (1.1),

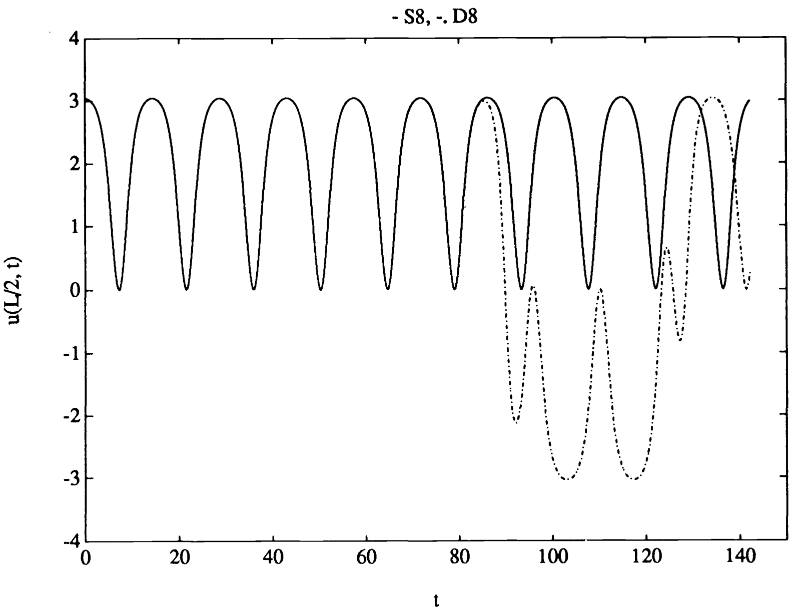


FIG. 7. *$u(L/2,t)$ against t.*

where the dependent variables $\mathbf{y}$ are the 32 discrete Fourier coefficients of the solution. This system of ordinary differential equations was integrated with the methods S8, Y8, S4, and D8 on $0 < t < 16L \approx 142.17$.

The standard method D8 was run with absolute error tolerances in the range 10^{-8} to 10^{-13}. Smaller tolerances were not tried because we felt that they would be too close to the size of the round-off error associated with the evaluation of the force $\mathbf{f}$ (this requires a couple of discrete Fourier transforms). None of the values of TOL we tried led to a successful integration, and D8 was not able to come up with the right qualitative behavior of the solution. The dash-dot line in Fig. 7 corresponds to TOL $= 10^{-13}$, $x = L/2$; the computed solution is completely wrong for $t > 80$. For this value of the tolerance the D8 code uses 32,810 function evaluations.

On the other hand, the symplectic algorithms S8, Y8, S4 were all able to identify the right qualitative behavior when run with suitable values of the step length h. With $h = \frac{1}{4}$ method S8 cannot faithfully describe the behavior of the solution up to the final time $t = 16L$. Halving the value of h to $h = \frac{1}{8}$ (27,312 evaluations) leads to a successful integration (see the curve in Fig. 7). For Y8, h has to be reduced down to $h = \frac{1}{16}$ (34,125 evaluations) and S4 requires $h = \frac{1}{32}$ (18,200 evaluations). Hence S4 was the most efficient, followed by S8 and Y8. Additional experiments show that if the final integration time is increased, further reduction of h is necessary to attain the correct qualitative behavior.

Following a referee's suggestion, we also integrated (7.1) and (7.2) in time by non-symplectic methods implemented with constant step sizes. The classical fourth-order Runge–Kutta method required $h = \frac{1}{64}$. This implies 36,400 function evaluations, which is more than any of the symplectic methods we tried. On the other hand, the order-8 formula of Dormand, El-Mikkawy, and Prince, when implemented with constant step sizes, was found to be able to identify the correct qualitative behavior. However, this needed $h = \frac{1}{64}$ and 72,800 function evaluations, i.e., more than twice as much computational effort, as the least efficient symplectic formula Y8.

Acknowledgments. The authors are grateful to J. de Frutos (Valladolid) for his help in programming the pseudospectral method and to R. Fletcher (Dundee) for his suggestions in connection with the continuation procedure. We also thank J. Dormand (Teeside), who kindly provided us with several of his RKN codes.

REFERENCES

[1] L. Abia and J. M. Sanz-Serna, *Partitioned Runge–Kutta methods for separable Hamiltonian problems*, Math. Comp., to appear.

[2] V. I. Arnold, *Mathematical Methods of Classical Mechanics*, 2nd ed., Springer-Verlag, New York, 1989.

[3] J. P. Calvo, *Métodos Runge–Kutta–Nyström Simplécticos*, Ph.D. thesis, Department of Applied Mathematics and Computation, University of Valladolid, Spain, 1992.

[4] M. P. Calvo and J. M. Sanz-Serna, *Variable steps for symplectic integrators*, in Numerical Analysis, 1991, D. F. Griffiths and G. W. Watson, eds., Longmans Press, London, 1992, pp. 32–48.

[5] ———, *Reasons for a failure. The integration of the two-body problem with a symplectic Runge–Kutta–Nyström code with stepchanging facilities*, in Applied Mathematics and Computation Reports, Report 1991/7, Universidad de Valladolid, Spain, 1991.

[6] ———, *Order conditions for canonical Runge–Kutta–Nyström methods*, BIT, 32 (1992), pp. 131–142.

[7] ———, *The development of variable-step symplectic integrators, with applications to the two-body problem*, SIAM J. Sci. Comput., 14 (1993), pp. 936–952.

[8] J. R. Dormand, M. E. El-Mikkawy, and J. P. Prince, *High-order embedded Runge–Kutta–Nyström formulae*, IMA J. Numer. Anal., 7 (1987), pp. 235–250.

[9] ———, *High-order embedded Runge–Kutta–Nyström formulae*, IMA. J. Numer. Anal., 7 (1987), pp. 423–430, corrigendum in 11 (1991), p. 297.

[10] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations* I, *Nonstiff Problems*, Springer-Verlag, Berlin (1987).

[11] B. M. Herbst and M. J. Ablowitz, *Numerical homoclinic instabilities in the Sine-Gordon Equation*, Quaestiones Mathematicae, 15 (1992), pp. 345–363.

[12] D. E. Knuth, *Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1973.

[13] D. Okunbor and R. D. Skeel, *An explicit Runge–Kutta–Nyström method is canonical if and only if its adjoint is explicit*, SIAM J. Numer. Anal., 29 (1992), pp. 521–527.

[14] ———, *Explicit canonical methods for Hamiltonian systems*, Math. Comp., 59 (1992), pp. 439–455.

[15] ———, *Canonical Runge–Kutta–Nyström Methods of Orders 5 and 6*, J. Comput. Appl. Math., to appear.

[16] J. M. Sanz-Serna, *The numerical integration of Hamiltonian systems*, in Computational Ordinary Differential Equations, J. R. Cash and I. Gladwell, eds., Clarendon Press, Oxford, 1992, pp. 437–449.

[17] ———, *Symplectic integrators for Hamiltonian problems: An overview*, in Acta Numerica 1992, Cambridge University Press, Cambridge, NY, 1992, pp. 243–286.

[18] J. M. Sanz-Serna and L. Abia, *Order conditions for canonical Runge–Kutta schemes*, SIAM J. Numer. Anal., 28 (1991), pp. 1081–1096.

[19] Y. B. Suris, *The canonicity of mappings generated by Runge–Kutta type methods when integrating the systems* $\ddot{x} = -\partial U/\partial x$, Zh. Vychisl. Mat. i Mat. Fiz., 29 (1989), pp. 202–211. (In Russian.); same as U.S.S.R. Comput. Math. and Math. Phys., 29 (1989) pp. 138–144.

[20] H. Yoshida, *Construction of higher order symplectic integrators*, Phys. Lett. A, 150 (1990), pp. 262–268.

SIAM J. SCI. COMPUT.
Vol. 14, No. 5, pp. 1253–1257, September 1993

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific 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 MAPPING ALGORITHM FOR PARALLEL SPARSE CHOLESKY FACTORIZATION*

ALEX POTHEN[†] AND CHUNGUANG SUN[‡]

Abstract. A task-to-processor mapping algorithm is described for computing the parallel multifrontal Cholesky factorization of irregular sparse problems on distributed-memory multiprocessors. The performance of the mapping algorithm is compared with the only general mapping algorithm previously reported. Using this mapping, the distributed multifrontal algorithm is nearly as efficient on a collection of problems with irregular sparsity structure as it is for the regular grid problems.

Key words. sparse Cholesky factorization, clique tree, distributed-memory multiprocessor, mapping algorithms, multifrontal method, parallel computing

AMS subject classifications. 65F50, 65F05

1. Introduction. Several implementations of parallel algorithms for sparse Cholesky factorization on distributed-memory multiprocessors have been described in the literature [1]–[3], [6], [8], [12]. Extensive discussions and bibliographies of previous work may be found in Liu's recent survey [11] of the multifrontal method, and in the Heath, Ng, and Peyton survey [9] of parallel sparse matrix factorization algorithms. Geist and Ng [4] have discussed a bin-pack mapping strategy for the fan-out algorithm suitable for mapping irregular problems. However, performance results for distributed sparse Cholesky factorization algorithms have been limited to regular model problems—e.g., the $n \times n$ nine-point grid ordered by optimal nested dissection [5] and a slightly less regular L-shaped problem. Most practical problems are far more irregular in structure than these model problems, and several algorithmic issues need to be addressed to obtain good performance on such problems. One important issue here is the mapping of tasks to processors to achieve load balance and low communication costs.

In this paper we describe a new task-to-processor mapping algorithm suitable for irregular sparse problems and apply it to the computation of a distributed multifrontal factorization. We compare the performance of the mapping algorithm with an adaptation of the sole previously reported general mapping algorithm—the bin-pack mapping algorithm [4]. We have experimented with a collection of problems with irregular sparsity structure and have obtained efficiencies comparable to those for the regular grid problems.

*Received by the editors August 27, 1991; accepted for publication (in revised form) January 21, 1993. This research was supported by National Science Foundation grant CCR-9024954 and by U.S. Department of Energy grant DE-FG02-91ER25095 at Pennsylvania State University.

†Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (apothen@narnia.uwaterloo.ca, na.pothen@na-net.ornl.gov).

‡Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, New York 14853-3801 (csun@cs.cornell.edu).

A multifrontal factorization is computed as a sequence of partial factorizations of a set of dense submatrices. We have chosen these dense submatrices to correspond to the maximal cliques from a clique tree representation of the Cholesky factor. Discussions of clique trees in the context of sparse matrix algorithms may be found in [10] and elsewhere. We have found the clique tree to be a convenient data structure for organizing the distributed multifrontal factorization. It also affords clear and concise descriptions and implementations of the distributed assembly and factorization algorithms. A detailed description of these algorithms is provided in [14], [16]. We denote the number of maximal cliques in the clique tree by m, the size of the clique tree (the sum of the number of vertices in the maximal cliques) by q, and the number of processors by ρ.

2. Mapping schemes. The *subtree-to-subcube mapping* scheme was proposed by George, Liu, and Ng [7] for the model grid problem. This scheme is only effective for problems with balanced tree structure and almost balanced workload distribution such as the model problem ordered by optimal nested dissection. Gilbert and Schreiber [8] use a two-dimensional bin-pack mapping strategy to allocate processors for factoring the frontal matrices at a stage in their CM-2 multifrontal factorization, and Ashcraft (personal communication) has considered a generalization of the subtree-to-subcube mapping scheme within his domain-separator model of Cholesky factorization.

Geist and Ng [4] have employed a bin-pack mapping scheme to map tasks associated with the elimination tree within the fan-out algorithm. We have adapted their scheme to map tasks associated with the clique tree for multifrontal factorization. In this scheme, the clique tree is explored beginning at the root until at least ρ vertex-disjoint subtrees are obtained. The subtrees are then packed into ρ bins using the *first-fit-decreasing* bin-packing heuristic. (In the heuristic, subtrees are processed in decreasing order of workloads, and a subtree is packed into the currently lightest bin.) The weight imbalance among the bins, α, is the ratio of the lightest bin to the heaviest bin. If α is larger than or equal to a user-specified tolerance τ, then the bin-pack mapping process terminates. Otherwise, the heaviest subtree in a bin is explored from its root and split further into subtrees, the ρ bins are repacked, and the ratio α recalculated and compared with τ. This process is repeated until $\alpha \geq \tau$, or the largest subtree cannot be split any further; in the latter case, we accept the imbalance among the bins.

The subtrees in a bin are assigned to a single processor; the cliques that lie on the path from the roots of these subtrees to the root of the clique tree remain to be mapped. We call such cliques *dominating cliques*. If a dominating clique K dominates a set of subtrees $T_1, \ldots, T_p$, then K is mapped to the subset of processors which compute these subtrees. This differs from the Geist–Ng scheme which wrap-maps the dominating cliques among all the processors. The latter scheme causes high communication costs and idle-waits in the factorization. The mapping algorithm can be implemented in $\mathcal{O}(m^2 \log m + q)$ time.

The *proportional mapping algorithm* may be viewed as a generalization of the subtree-to-subcube mapping in which both the structure and the workload distribution of the tree are taken into consideration. A variant of this mapping has been considered in the context of distributed sparse orthogonal factorization and the fan-in and fan-out algorithms [15]. The algorithm is described in Fig. 1. The variable $tree(K)$ denotes the set of subtrees rooted at the child cliques of a clique K, $proc(T)$ the set of processors allocated to a subtree T, and $w(T)$ the workload associated with a subtree T.

Many details are important in obtaining a correct and effective proportional mapping algorithm since rounding to integral numbers of processors causes several problems. The algorithm processes the subtrees of a clique in decreasing order of their workloads to avoid unnecessary communication. Rounding may cause unallocated processors to remain after an initial mapping of processors to the subtrees of a clique; it also may

```
prop_map($\widehat{T}$: clique tree, $P$: set of processors)
$S := P, \rho := |P|$;
if ($\rho = 1$) then
    map the subtree $\widehat{T}$ to the only processor in $P$;
else
    let $K$ be the root of $\widehat{T}$;   wrap-map the columns of $K$ to processors in $P$;
    if $K$ is a leaf then return;
    {process the subtrees of $K$ one by one}
    $w := \sum_{T \in tree(K)} w(T)$;
    while not all subtrees in $tree(K)$ are processed do
        choose a heaviest subtree $T$ in $tree(K)$ not yet processed;
        $k = \lfloor (w(T)/w) * \rho + 0.5 \rfloor$;
        if $k = 0$ or $S = \emptyset$ then
            find a processor $p \in P$ with the least workload;   $proc(T) := \{p\}$;
        else
            if $k > |S|$ then $k := |S|$;
            $proc(T) :=$ a set of $k$ processors in $S$;   $S := S \setminus proc(T)$;
        end if
    end while
    {allocate processors which may be left unallocated}
    while $S \neq \emptyset$ do
        let $p$ be a processor in $S$;   $S := S \setminus \{p\}$;
        find a subtree $T$ in $tree(K)$ such that
        processors computing $T$ have the highest workload;
        $proc(T) := proc(T) \cup \{p\}$;
    end while
    {map the subtrees of $K$ recursively}
    for each subtree $T$ in $tree(K)$ do prop_map($T, proc(T)$);
end if
```

FIG. 1. *The proportional mapping algorithm.*

cause unmapped subtrees to remain after all processors have been allocated in an initial mapping. Our algorithm deals with all these issues; its time complexity is $O(m \log m + q)$.

3. Computational results. The performance of the distributed multifrontal algorithm was tested on the model problem and on a set of problems chosen from the Harwell–Boeing and the NASA Ames collections. Preprocessing steps were done sequentially. All programs were written in C, and run on an Intel iPSC/2 multiprocessor using double precision arithmetic. Version 3.2 of the the iPSC/2 UNIX operating system was employed, and the default compiler optimization was used.

The characteristics of the test problems are shown in Table 1. In this table, n is the order of a matrix, $|A|$ ($|L|$) is the number of nonzeros in the lower triangle of the matrix A (L), m is the number of maximal cliques in the clique tree, and "op_count" is the total number of flops.

For each problem, tolerances of 40%, 60%, and 80% have been used to test the bin-pack mapping algorithm. The model nine-point grid problem is ordered by optimal nested dissection (ND), and all other problems are ordered by the multiple minimum degree (MMD) ordering. Running times are shown in Table 2; times for the subtree-to-subcube mapping for irregular problems are not shown because it performs much worse than the other mappings.

TABLE 1
Characteristics of the test problems.

Problem	n	$\lvert A\rvert$	$\lvert L\rvert$	m	op_count
GRID127	16,129	79,885	518,578	8,191	38,387,343
BCSPWR10	5,300	13,571	28,064	4,846	332,784
BCSSTK13	2,003	42,924	271,671	597	59,991,546
BCSSTK21	3,600	15,100	90,454	2,400	4,449,886
LSHP3466	3,466	13,681	86,582	1,845	4,299,631
NASA2146	2,146	37,198	135,798	306	1,187,5021
NASA4704	4,704	54,730	281,472	1,241	36,008,716

TABLE 2
Running times (seconds) obtained on the test problems.

Problem	p	prop	b40	b60	b80	sub
GRID127 (ND)	4	74.22	74.14	74.14	74.14	74.22
	8	43.22	43.14	43.14	43.09	43.23
	16	24.82	24.64	33.52	63.50	24.80
	32	14.45	14.38	19.94	41.63	14.64
BCSPWR10 (MMD)	1	4.078				
	2	2.976	2.981	2.321	2.321	
	4	2.187	1.588	1.390	1.529	
	8	1.362	1.080	1.061	1.342	
	16	0.902	1.053	1.233	1.390	
	32	0.854	1.316	1.380	1.536	
BCSSTK13 (MMD)	2	241.6	241.6	241.6	241.9	
	4	142.6	199.8	187.7	187.7	
	8	75.64	108.3	109.8	108.8	
	16	42.92	58.64	59.24	70.72	
	32	27.15	43.23	44.42	44.42	
BCSSTK21 (MMD)	1	35.59				
	2	17.82	17.84	17.84	17.84	
	4	9.376	17.97	17.97	17.97	
	8	6.541	14.43	13.92	9.119	
	16	4.015	8.514	9.397	10.71	
	32	2.811	7.117	7.583	7.690	
LSHP3466 (MMD)	1	33.67				
	2	17.63	17.64	17.64	17.64	
	4	11.16	13.09	10.81	11.76	
	8	6.642	7.595	7.244	7.251	
	16	4.138	7.467	8.734	7.437	
	32	3.780	5.298	6.725	7.168	
NASA2146 (MMD)	1	87.89				
	2	59.92	59.89	47.61	47.61	
	4	27.91	30.19	32.87	33.31	
	8	16.60	15.98	15.98	34.47	
	16	9.290	15.63	19.25	18.78	
	32	7.375	10.73	12.60	14.82	
NASA4704 (MMD)	2	159.4	140.0	140.0	140.0	
	4	88.49	89.02	101.7	101.7	
	8	42.05	41.59	49.71	62.72	
	16	27.56	38.21	45.37	39.68	
	32	16.62	25.65	35.03	36.23	

The results clearly show that proportional mapping performs better than the bin-pack mapping as the number of processors increases. Furthermore, the performance of the bin-pack mapping is sensitive to the tolerance used, and one value of the tolerance is not uniformly better than another over varying the number of processors for a given problem. This is a serious drawback since it is difficult to determine an appropriate tolerance a priori.

For the NASA4704 problem when 32 processors are used the efficiency is 0.60; on the other hand, for BCSPWR10 the efficiency is only 0.15 when 32 processors are used. These results are a consequence of two facts. First, on the iPSC/2 the cost of communicating data is high relative to the cost of floating point work. Second, the number of elements communicated is asymptotically a lower-order term than the number of floating point operations performed. Hence efficiencies are high only when there is a sufficient amount of arithmetic work per processor for the arithmetic costs to dominate the communication and data structure overhead costs.

Another factor that influences efficiency is the balance of the clique tree. Well-balanced clique trees are mapped more efficiently since, for such trees, rounding to integral numbers of processors creates less load imbalance. Optimal nested dissection of the model problem creates well-balanced trees, while the MMD ordering of the less regular problems leads to less balanced trees. The effect of recent orderings with low fill and good balance such as spectral nested dissection [13] remains to be evaluated.

REFERENCES

[1] C. Ashcraft, *The aggregate/element model for distributed column-based Cholesky factorization*, Tech. Report ECA-TR-160, Boeing Computer Services, Seattle, WA, March 1991.

[2] C. Ashcraft, S. C. Eisenstat, and J. W. H. Liu, *A fan-in algorithm for distributed sparse numerical factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 593–599.

[3] C. Ashcraft, S. C. Eisenstat, J. W. H. Liu, and A. H. Sherman, *A comparison of three column-based distributed sparse factorization schemes*, Tech. Report CS-90-09, Computer Science, York University, North York, Ontario, Canada, August 1990.

[4] G. A. Geist and E. G. Y. Ng, *Task scheduling for parallel sparse Cholesky factorization*, Internat. J. Parallel Programming, 18 (1989), pp. 291–314.

[5] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[6] J. A. George, M. T. Heath, J. W. H. Liu, and E. G. Y. Ng, *Solution of sparse positive definite systems on a hypercube*, J. Comput. Appl. Math., 27 (1989), pp. 129–156.

[7] J. A. George, J. W. H. Liu, and E. G. Y. Ng, *Communication results for parallel sparse Cholesky factorization on a hypercube*, Parallel Comput., 10 (1989), pp. 287–298.

[8] J. R. Gilbert and R. S. Schreiber, *Highly parallel sparse Cholesky factorization*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 1151–1172.

[9] M. T. Heath, E. G. Y. Ng, and B. W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Rev., 33 (1991), pp. 420–460.

[10] J. G. Lewis, B. W. Peyton, and A. Pothen, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1146–1173.

[11] J. W. H. Liu, *The multifrontal method for sparse matrix solution: theory and practice*, SIAM Rev., 34 (1992), pp. 82–109.

[12] R. Lucas, T. Blank, and J. Tiemann, *A parallel solution method for large sparse systems of equations*, IEEE Trans. Computer-Aided Design, CAD-6 (1987), pp. 981–991.

[13] A. Pothen, H. D. Simon, and L. Wang, *Spectral nested dissection*, Tech. Report 92-01, Computer Science, Pennsylvania State University, University Park, PA, 1992.

[14] A. Pothen and C. Sun, *A distributed multifrontal algorithm using clique trees*, Tech. Report 91-24, Computer Science, Pennsylvania State University, University Park, PA, September 1991.

[15] P. Raghavan, *Distributed Sparse Matrix Factorization: QR and Cholesky Decompositions*, Ph.D. thesis, Pennsylvania State University, University Park, PA, 1992.

[16] C. Sun, *Some Applications of Clique Trees to Sparse Cholesky Factorizations*, Ph.D. thesis, Pennsylvania State University, University Park, PA, 1992.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1259–1288, November 1993

BLOCK-CYCLIC DENSE LINEAR ALGEBRA*

WOODY LICHTENSTEIN† AND S. LENNART JOHNSSON†‡

Abstract. Block-cyclic order elimination algorithms for LU and QR factorization and solve routines are described for distributed memory architectures with processing nodes configured as two-dimensional arrays of arbitrary shape. The cyclic-order elimination, together with a consecutive data allocation, yields good load balance for both the factorization and solution phases for the solution of dense systems of equations by LU and QR decomposition. Blocking may offer a substantial performance enhancement on architectures for which the level-2 or level-3 BLAS (basic linear algebra subroutines) are ideal for operations local to a node. High-rank updates local to a node may have a performance that is a factor of four or more higher than a rank-1 update.

This paper shows that in many parallel implementations, the $O(N^2)$ work in the factorization may be of the same significance as the $O(N^3)$ work, even for large matrices. The $O(N^2)$ work is poorly load balanced in two-dimensional nodal arrays, which are shown to be optimal with respect to communication for consecutive data allocation, block-cyclic order elimination, and a simple, but fairly general, communications model.

In this Connection Machine system CM-200 implementation, the peak performance for LU factorization is about 9.4 Gflops/s in 64-bit precision and 16 Gflops/s in 32-bit precision. Blocking offers an overall performance enhancement of an approximate factor of two. The broadcast-and-reduce operations fully utilize the bandwidth available in the Boolean cube network interconnecting the nodes along each axis of the two-dimensional nodal array embedded in the cube network.

Key words. linear algebra, distributed memory, LU, QR, cyclic order

AMS subject classifications. 15A23, 65F05, 68-04, 68N99, 68Q25

1. Introduction.

The main contributions of this paper are: (i) empirical evidence that a block-cyclic order elimination can be used effectively on distributed memory architectures to achieve load balance as an alternative to block-cyclic data allocation; (ii) a discussion of the issues that arise when the block-cyclic orderings of rows and columns are different, which is the typical case when the number of processing nodes is not a square; and (iii) a proof that within a wide class of regular data layouts, two-dimensional nodal arrays with consecutive (block) data allocation and cyclic elimination order are optimal for elimination-based dense linear algebra routines. This last result applies to communication systems in which the communication time is a function only of the number of elements entering or leaving a node. The effectiveness of the block-cyclic order elimination demonstrates the utility of equivalencing block-distributed and block-cyclic distributed arrays in Fortran D [7] and Vienna Fortran [31].

The programs described in this article were written for an implementation of the Connection Machine Scientific Software Library (CMSSL) [27] on the Connection Machine system CM-200 [28]. This system is a distributed memory computer with up to 2048 nodes. Each node has hardware support for floating-point addition and multiplication in 32-bit and 64-bit precision. Each node has up to 4 Mbytes of local memory, a single 32-bit wide data path between the floating-point processor and the local memory, and separate communication circuitry. Data paths internal to the floating-point unit are 64 bits wide. The processing units are interconnected as an 11-dimensional Boolean cube, with a pair of channels between adjacent nodes. Data may be exchanged on all 22 (11 × 2) channels of every node concurrently. This property is exploited for data copying (spread) and data summation (reduction) in the algorithms described below.

*Received by the editors January 29, 1992; accepted for publication October 6, 1992.

†Thinking Machines Corporation, 245 First Street, Cambridge, Massachusetts 02142-1214 (woody@think.com, Johnsson@think.com).

‡This author is also affiliated with the Division of Applied Sciences, Harvard University, Cambridge, Massachusetts 02138.

We consider LU factorization, QR factorization (with and without partial pivoting), and solution routines for both LU factorization (triangular system solvers) and QR factorization. Our triangular solver encompasses both the routine _TRSV in the level-2 BLAS (basic linear algebra subroutines) [6] and the routine _TRSM in the level-3 BLAS [4], [5]. It is easy to show that a cyclic data allocation with consecutive order elimination, or a consecutive allocation and a cyclic order elimination, yields a factor of three higher processor utilization on two-dimensional nodal arrays than consecutive allocation and consecutive elimination order [17]. Coleman [22], [23] reports some results from an implementation of triangular system solvers using consecutive order elimination and cyclic data allocation on multiprocessors with up to 128 nodes. Van de Geijn [30] uses consecutive order elimination and cyclic data allocation for an implementation of LU factorization and triangular system solvers. Since the Connection Machine system compilers by default use consecutive data allocation, we use a cyclic-order elimination. We use a similar implementation strategy for routines for reduction of a symmetric or Hermitian matrix to real tridiagonal form (EISPACK_TRED and _HTRID [26], LAPACK_SYTRD and _HETRD [1]). Details concerning these routines will appear elsewhere.

We use blocking of row and column operations to increase the efficiency of operations in each node. The level-2 BLAS is used in each node to achieve maximum performance. The difference in peak performance between a rank-1 update and a higher-rank update is about a factor of four on a Connection Machine system CM-200 node. The difference in peak performance is mostly determined by the difference in need for memory bandwidth between a rank-1 and a high rank update. In our CMSSL implementation of the BLAS local to each Connection Machine system CM-200 node [20], LBLAS, each node achieves a peak performance of about 9.3 Mflops/s in 64-bit precision on matrix multiplication (high rank updates). Our implementation of LU factorization of matrices distributed over all nodes achieves a peak performance, including communication, of 4.6 Mflops/s per node in 64-bit precision. As a comparison, our CMSSL implementation of dense matrix multiplication with operands distributed across all nodes achieves a peak performance of 4.8 Gflops/s in 64-bit precision [24].

This article provides sufficient insight into the details of the algorithms to account for the difference in performance between local matrix multiplication and global factorization and solves routines for dense matrices. We also describe how the performance scales with problem and machine size. In §2 we review the merits of higher-level local BLAS for a class of common processor architectures. Section 3 discusses the performance of the level-1 and level-2 LBLAS used in CMSSL. Section 4 describes the layouts of data arrays created by the Connection Machine Run-Time System. Section 5 explains how we use block-cyclic ordering of the elimination steps to keep the work load balanced across all nodes. Section 6 discusses the performance characteristics of the different parts of the algorithms. Finally, we show in §7 that a two-dimensional consecutive data layout is optimal for elimination algorithms using a cyclic order elimination and communication systems where the times for data copying and reduction are determined by the number of data items leaving or entering a node.

2. Blocking for improved performance of local BLAS.

Memory bandwidth is the most critical resource in high-performance architectures. Therefore, proper attention must be given to the primitives used in constructing linear algebra libraries. Nearly all floating-point computation in linear algebra occurs as multiply-add pairs. This fact is evident in the BLAS used to perform many operations in the solution of linear systems of equations, eigenanalysis, optimization, and the solution of partial differential equations. The first BLAS were vector routines (like DSCAL and DDOT) [21]. But, these routines require large memory bandwidth for peak floating-point performance, and algorithm designers turned to higher-level BLAS, such as level-2 BLAS for matrix-vector operations, and level-3 BLAS for matrix-matrix operations. Next, we give a few examples to illustrate this fact.

Today, most high-performance floating-point processors have the ability to perform concurrently one multiplication and one addition. The data for these operations is nearly always read from registers, and the results are written to registers. The peak performance can only be achieved when multiplication and addition can be performed concurrently, and the data they require can flow in and out of registers fast enough. The memory bandwidth required to match the computational bandwidth depends on the computation being performed.

For example, the DSCAL operation, which multiplies a vector by a scalar with all operands in 64-bit precision, reads one scalar into a register, and then, for each component of the result, reads one element of the argument vector, multiplies it by the constant, and writes one result to memory. Thus, each multiplication requires 8 bytes to be loaded into a register and 8 bytes to be stored from a register, or 16 bytes of memory bandwidth per floating-point operation. As a contrast, the DDOT routine, which computes the inner product of two vectors in 64-bit precision, reads two 8-byte quantities for each multiply-add it performs, then stores one 8-byte result at the end. Thus 16 bytes of memory bandwidth is required per two floating-point operations, or 8 bytes per floating-point operation. Similarly, it is easy to show that the rank-1 update routine DGER requires a memory bandwidth of 8 bytes per floating-point operation.

Matrix-vector multiplication, performed by the routine DGEMV, can be organized such that the vector is read into registers, and the matrix-vector multiplication computed through the accumulation of scaled vectors as $y \leftarrow \alpha x + y$. With α and y allocated to registers, and x representing a column of the matrix read from memory, 8 bytes of memory bandwidth is required for every pair of floating-point operations in 64-bit precision.

A level-3 BLAS routine such as DGEMM allows for a further reduction in memory bandwidth requirement. It performs the operation $C \leftarrow A \times B$, which may be performed as a sequence of operations on $b \times b$ subblocks. If the blocks fit into the registers, then $2b^3$ floating-point operations may be computed using $3b^2$ input elements (b^2 elements per operand), producing b^2 results. If all contributions to a block of C are accumulated in registers, then it suffices to load $2b^2$ inputs for each set of $2b^3$ floating-point operations. All stores are delayed until all computations for a $b \times b$ block of C are completed. Therefore, $16b^2/2b^3$ bytes of memory bandwidth are required per floating-point operation in 64-bit precision, or $8/b$ bytes/flop. A high-rank update of a matrix is equivalent to matrix multiplication.

Table 1 summarizes the memory bandwidth requirements for a subset of the BLAS. The significance of the difference in memory bandwidth requirement of the different routines depends upon the available memory bandwidth. For example, a computer with three 8-bytes wide data paths between each processor and the memory, i.e., a 12-bytes/flop computer such as the Cray-YMP, can perform DAXPY operations at peak rates. A 2-bytes/flop computer, such as an Intel i860 [10] and a Connection Machine system CM-200 node, may not have enough registers to achieve peak rates even for level-3 BLAS local to each node, such as for the DGEMM routine. As a rule of thumb, the less memory bandwidth is available, the more blocking is desirable, but more blocking is only useful if there is a sufficient number of registers.

This simplified performance picture is in reality often complicated by pipeline delays and looping overheads. For short vectors and small register sets, minimizing these

TABLE 1
Memory bandwidth requirement for full utilization of a floating-point unit with one adder and one multiplier.

Operation	Memory bandwidth bytes/flop
DSCAL	16
DAXPY	12
DDOT	8
DGER	8
DGEMV	4
DGEMM	$8/b$

quantities may be as important for performance as minimizing the demand for memory bandwidth.

3. Local BLAS on the Connection Machine system CM-200. In the following, we refer to the BLAS local to each node as LBLAS to distinguish it from BLAS for data arrays distributed over several nodes, DBLAS. Each node of the Connection Machine system CM-200 is a 2-bytes/flop computer, based on the memory bandwidth. There is a single 32-bit wide data path between each floating-point processor and its local memory. The data paths internal to a processor are 8 bytes wide (i.e., internally each floating-point processor is a 4-bytes/flop computer). Each floating-point processor has 32 floating-point registers and operates at 10 MHz. There is a one-cycle delay for loading data from memory and a three-cycle arithmetic pipeline delay. Each vector operation incurs an overhead of at least six cycles. Moreover, the memory is organized into pages, and a page fault incurs a delay of one cycle. Stores are relatively more expensive and require close to two cycles. The floating-point processor can achieve close to peak performance in 32-bit precision for level-2 LBLAS with the operands local to a node and at least half of the peak performance in 64-bit precision. The floating-point processor does not have enough registers to achieve its full peak rate on level-3 LBLAS. Table 2 gives the actual performance for the SAXPY, DAXPY, SDOT, and DDOT routines as a function of the vector length for *each* Connection Machine system CM-200 node. Tables 3 and 4 give the performance for SGEMV and DGEMV as a function of matrix size.

Figure 1 shows the performance for the DAXPY and DDOT routines on each Connection Machine system CM-200 node as a function of the vector length. The figure also shows the performance of the routine DGEMV as a function of the number of columns for 64 matrix rows. The peak measured performance for the _AXPY routine is about 60% of the peak performance of the _DOT routine. In 32-bit precision the performance is comparable up to a vector length of about 30, while for 64-bit precision the difference is measurable even for short vectors. Note that the _AXPY routine requires twice the memory bandwidth of the _DOT routine. The performance of the _GEMV routine is about twice that of the _DOT routine. This behavior is expected, since the memory bandwidth requirement of the _GEMV routine is about half of that of the _DOT routine.

Our implementation of the rank-1 update makes use of _AXPY operations, while the higher-rank updates are based on matrix-vector or vector-matrix multiplication, depending upon the shape of the operands [20].

Given the performance characteristics of the LBLAS on the Connection Machine system CM-200, it is desirable to base linear algebra algorithms on the level-2 LBLAS. The potential performance gain from blocking the operations on b rows and columns is illustrated in Table 5. This table shows the speedup of rank-b updates relative to a rank-1

TABLE 2
*Level-*1 LBLAS *execution rates in* Mflops/s *on* each *Connection Machine system* CM-200 *floating-point processor.*

V-length	SAXPY	DAXPY	SDOT	DDOT
1	0.21	0.16	0.21	0.16
2	0.41	0.29	0.35	0.35
3	0.58	0.42	0.52	0.51
4	0.80	0.54	0.68	0.66
5	0.97	0.65	0.83	0.79
8	1.40	0.95	1.27	1.15
10	1.65	1.13	1.53	1.36
15	2.15	1.41	2.13	1.79
20	2.55	1.61	2.65	2.13
30	3.11	1.87	3.03	2.21
40	3.07	1.89	3.71	2.70
50	3.37	2.02	4.23	2.96
60	3.53	2.09	4.41	3.02
70	3.56	2.11	4.75	3.19
80	3.71	2.18	5.06	3.33
100	3.77	2.22	5.32	3.45
120	3.93	2.28	5.51	3.53
140	4.00	2.31	5.86	3.68
256	4.20	2.39	6.52	3.92
512	4.32	2.43	6.97	4.07
1024	4.42	2.46	7.24	4.15
2048	4.46	2.48	7.38	4.20
4096	4.49	2.49	7.45	4.22
8192	4.50	2.49	7.49	4.24

TABLE 3
Execution rate in Mflops/s *for matrix-vector multiplication on* each *Connection Machine system* CM-200 *floating-point processor.*

SGEMV											
Number of rows	Number of columns										
	2	3	4	5	8	16	32	64	128	512	2048
2	0.62	0.98	1.23	1.46	2.46	4.03	5.93	7.38	8.62	9.90	10.30
3	0.99	1.39	1.74	2.05	3.38	6.03	8.21	9.80	10.8	11.9	12.20
4	1.35	1.88	2.33	2.73	4.40	6.72	8.87	10.70	12.00	13.10	13.50
5	1.62	2.24	2.77	3.24	5.11	8.43	10.6	12.20	13.20	14.10	14.40
8	2.33	3.19	3.91	4.52	6.80	9.47	11.90	13.70	14.70	15.70	15.90
16	3.68	4.91	5.90	6.71	9.27	12.10	14.40	15.80	16.70	17.40	17.50
32	4.29	5.96	6.39	7.56	10.20	12.90	14.90	16.10	16.80	17.40	17.50
64	5.13	6.85	7.61	8.71	11.50	14.00	15.80	16.80	17.40	17.90	—
128	5.51	7.09	8.01	8.94	11.90	14.40	16.00	17.00	17.50	17.90	—
512	5.80	7.30	8.32	9.15	12.10	14.40	15.90	16.80	17.30	—	—
2048	5.92	7.40	8.45	9.25	12.10	14.30	15.70	—	—	—	—

update of a 32×32 matrix and a 512×512 matrix local to a Connection Machine system CM-200 node.

Blocking yields a larger relative performance gain for small matrices than for big matrices. For large submatrices per node a performance gain by a factor in excess of 3.5 is possible through the use of high rank updates. About 90% of this gain is achieved for a blocking factor of 16. For relatively small submatrices per node a performance gain in excess of a factor of 4.5 is possible through the use of high rank updates. About 85% of this gain is achieved by the use of a blocking factor of 16.

TABLE 4
Execution rate in Mflops/s *for matrix-vector multiplication on* each *Connection Machine system* CM-200 *floating-point processor.*

DGEMV											
Number of rows	Number of columns										
	2	3	4	5	8	16	32	64	128	512	2048
2	0.64	0.89	1.10	1.29	2.03	3.04	3.93	4.67	5.17	5.62	5.74
3	0.91	1.25	1.53	1.78	2.71	4.20	5.15	5.80	6.21	6.56	6.66
4	1.13	1.54	1.88	2.17	3.22	4.44	5.51	6.26	6.72	7.12	7.23
5	1.33	1.80	2.18	2.51	3.64	5.23	6.21	6.84	7.22	7.54	7.63
8	1.80	2.39	2.87	3.26	4.47	5.82	6.85	7.51	7.90	8.11	8.22
16	2.54	3.29	3.79	4.25	5.58	6.89	7.80	8.35	8.65	8.90	8.97
32	2.79	3.60	4.11	4.69	5.92	7.14	7.95	8.44	8.70	8.92	—
64	3.19	4.10	4.60	5.12	6.37	7.52	8.26	8.69	8.93	9.11	—
128	3.33	4.20	4.74	5.22	6.49	7.60	8.30	8.70	8.92	—	—
512	3.45	4.33	4.86	5.30	6.53	7.57	8.22	8.60	—	—	—
2048	3.49	4.89	4.91	5.34	6.57	7.60	—	—	—	—	—

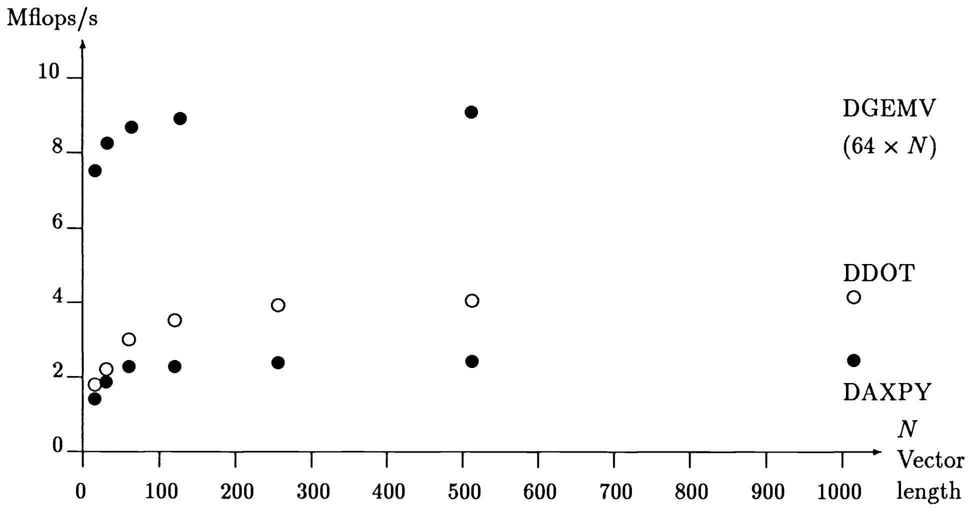


FIG. 1. *The execution rates in* Mflops/s *of the* DAXPY, DDOT, *and* DGEMV LBLAS *on* each *Connection Machine system* CM-200 *node.*

TABLE 5
*Speedup of rank-b updates over rank-*1 *updates for* each *Connection Machine system* CM-200 *node.*

Rank	Matrix shape	
	32 × 32	512 × 512
1	1.00	1.00
2	1.49	1.42
4	2.20	2.00
8	3.17	2.69
16	3.82	3.12
32	4.25	3.38
64	4.51	3.54

4. Standard data layouts. The routines described in this article are designed to operate in place on arrays passed to the routines from high-level language programs. The data motion requirements and the performance depend strongly upon the data allocation of the operands. The default allocation of data arrays to nodes determined by the Connection Machine Run-Time System is based entirely on the shape of the data arrays. Each array is by default distributed evenly over all nodes, i.e., the Connection Machine systems support a global address space. In the default array allocation mode, the nodes are configured for each array such that the number of axes in the data array and in the nodal array is the same. The ordering of the axes is also the same. When there are more matrix elements than nodes, consecutive elements along each data array axis (a block) are assigned to a node. The ratios of the lengths of the axes of the nodal array are approximately equal to the ratios between the lengths of the axes of the data array [29]. In such a configuration the lengths of the local segments of all axes are approximately the same, and the communication needs are minimized when references along the different axes are equally frequent. The default array layout is known as a *cannonical layout.* In §7 we show that the canonical layout is optimum for LU and QR factorization for a simple, but realistic, communications model. In [24] we show that the canonical layout is also optimal for matrix multiplication.

The canonical layout can be altered through compiler directives. An axis can be forced to be local to a node by the directive SERIAL, if there is sufficient local memory. The length of the local segment of an axis can also be changed by assigning *weights* to the axes. High weights are used for axes with frequent communication, and low weights for axes with infrequent communication. A relatively high weight for an axis increases the length of the local segment of that axis, at the expense of the lengths of the segments of the other axes. The total size of the subarray is independent of the assignment of weights. Only the shape of the subarray changes. The shape of the nodal array is important, since it affects the performance of global operations such as data copying, data summation, and pivot selection. The nodal array shape is also important for the performance of the LBLAS, since it affects the shape of the subarrays assigned to each node and, hence, vector lengths and the relative importance of loop overhead.

In many computations more than one array is involved, and the relative layouts of the arrays may be important. For instance, in solving a linear system of equations there are two arrays involved when the computed solutions overwrite the right-hand sides. The required communication in the triangular systems solver depends in a significant way on how the triangular factors and the right-hand sides are allocated to the nodes. With the original matrix factored in place, the triangular factors are stored in a two-dimensional data array, for which the default nodal array shape is a two-dimensional array. For a single right-hand side, the default nodal array shape is a one-dimensional array. Even if there are many right-hand sides, there is no guarantee that the shapes of the nodal arrays are the same for the data array to be factored and the data array of right-hand sides. The ALIGN compiler directive may be used to assure that different data arrays are assigned to nodes using the same nodal array shape for the allocation.

The consecutive allocation scheme [17] selects elements to be assigned to the same node. Compiler directives, such as axis weights, SERIAL and ALIGN, address the issue of choosing the nodal array shape. Another data layout issue is the assignment of data sets to nodes, where a set is made up of consecutive elements along each data array axis. The network topology and the data reference pattern are two important characteristics in this assignment. The nodes of the Connection Machine system CM-200 are interconnected as a Boolean cube with up to 11 dimensions. A Boolean cube network of n dimensions has 2^n nodes.

The nodes of a Boolean cube can be given addresses such that adjacent nodes differ in precisely one bit in their binary encoding. Assigning subarrays to nodes using the standard binary encoding of the subarray index along an axis does not preserve adjacency along an axis. For instance, three and four differ in all three bits in the encoding of the addresses of eight nodes and are at a distance of three apart. In general, $2^{n-1} - 1$ and 2^{n-1} differ in n bits in their binary encoding and are at a distance of n. The number of bits in which two indices differ translates directly into distance in a Boolean cube architecture.

Binary-reflected Gray codes [25] generate embeddings of arrays into Boolean cube networks that preserve adjacency [17]. Gray codes have the property that the encoding of successive integers differ in precisely one bit. In a Boolean cube network successive indices are assigned to adjacent nodes. The binary-reflected Gray code is efficient, both in preserving adjacency and in node utilization, when the length of the axes of the data array is a power of two [11]. For arbitrary data array axes' lengths, the Gray code may be combined with other techniques to generate efficient embeddings [3], [12].

The binary-reflected Gray code embedding is the default embedding on the Connection Machine system CM-200, and is enforced by the compiler directive NEWS for each axis. The standard binary encoding of each axis is obtained through the compiler directive SEND. The performance of global operations, such as pivot selection and broadcast required in the factorization and solve routines, is insensitive to the data distribution along each axis. Our factorization and solver routines also require permutations for rectangular nodal arrays. These permutations may exhibit some sensitivity to whether arrays are allocated in NEWS or SEND order.

5. Cyclic-order factorization and triangular system solution. We consider the solution of a system of equations $AX = Y$ through factorization and forward and backward substitution. We consider both single and multiple right-hand sides. We present algorithms for load-balanced factorization and triangular systems solution on data arrays allocated to nodes by a consecutive data allocation scheme. The matrix is factored in place, and the solutions overwrite the right-hand sides.

5.1. Factorization.

5.1.1. Balanced work load.

For an $N \times N$ matrix assigned to a $p \times q$ nodal array, each node is assigned a submatrix of shape $N/p \times N/q$. With the consecutive allocation scheme, the submatrix on node (i,j), $1 \leq i \leq p$, $1 \leq j \leq q$ consists of elements (a,b) with $(i-1) * N/p < a \leq i * N/p$ and $(j-1) * N/q < b \leq j * N/q$.

An LU factorization algorithm steps through the rows of a matrix, subtracting a multiple of a row, the pivot row, from all rows not yet selected as pivot rows. If the matrix is allocated to the nodes with a consecutive allocation scheme, and the pivot rows are selected in order, then after N/p rows have been chosen, the first row of nodes will no longer have any work to do. One method for avoiding this imbalance is to redistribute the rows and columns of the matrix periodically, as the work load becomes unbalanced. This approach leads to substantial communications overhead. The characteristics of QR decomposition are similar, though instead of subtracting the pivot row from rows not yet selected as pivots, a normalized linear combination of those rows is subtracted from each of them.

A more elegant technique is to use a cyclic data allocation scheme [8], [17], [22], [23], [30]. In a cyclic data allocation scheme the first row of the matrix is placed on the first row of nodes, the second row of the matrix on the second row of nodes, and so on until one matrix row has been assigned to each of the p rows of nodes. Then, the $(p+1)$st matrix row is assigned to the first row of nodes, etc. With this method, matrix rows are consumed evenly from the different rows of nodes during elimination, so that no row of nodes ever has more than one more active matrix row than any other row of nodes. Columns are treated similarly.

The space (rows and columns) and time (pivot selection) dimensions are interchangeable [13]–[16]. Hence instead of distributing rows and columns cyclically over the two-dimensional nodal array, pivot rows and columns can be selected cyclically to achieve the desired load balance. We have chosen this approach for our implementation. To allow for the use of level-2 LBLAS, blocking of rows and columns on each node is used. In LU factorization a blocking of the operations on b rows and columns means that b rows are eliminated at a time from all the other rows.

In summary, we factor a matrix in place using a *block-cyclic elimination order* for both rows and columns. For block size b we first eliminate rows $1, \ldots, b$, then rows $(p+1), \ldots, (p+b)$, etc. We are eliminating b matrix rows at a time from each row of nodes. A block-cyclic elimination order was recommended in [15] for load-balanced solution of banded systems.

Below we give an example of a block-cyclic ordered elimination for a square array of nodes and a block size of two. Pivot rows and columns are indicated by numbers, eliminated elements by periods, and all other elements are shown as an asterisk.

Example I: $N = 16, p = q = 4, b = 2$.

$$\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 1.}}\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
1&1&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast
\end{array}\right]
\quad
\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 2.}}\\
\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
.&.&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&2&2&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast
\end{array}\right]$$

$$\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 3.}}\\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\[1ex]
. & . & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\[1ex]
. & . & 3 & 3 & . & . & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3\\
. & . & 3 & 3 & . & . & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\[1ex]
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & 3 & 3 & * & * & * & * & * & *
\end{array}\right]
\quad
\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 4.}}\\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\[1ex]
. & . & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\[1ex]
. & . & * & * & . & . & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\[1ex]
. & . & 4 & 4 & . & . & 4 & 4 & . & . & 4 & 4 & 4 & 4 & 4 & 4\\
. & . & 4 & 4 & . & . & 4 & 4 & . & . & 4 & 4 & 4 & 4 & 4 & 4\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & 4 & 4 & * & *
\end{array}\right]$$

$$\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 5.}}\\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & 5 & 5 & . & . & 5 & 5 & . & . & 5 & 5 & . & . & 5 & 5\\
. & . & 5 & 5 & . & . & 5 & 5 & . & . & 5 & 5 & . & . & 5 & 5\\[1ex]
. & . & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & * & * & * & * & * & * & * & * & * & * & *\\
. & . & 5 & 5 & . & . & * & * & . & . & * & * & . & . & * & *\\
. & . & 5 & 5 & . & . & * & * & . & . & * & * & . & . & * & *\\[1ex]
. & . & * & * & . & . & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & * & * & * & * & * & * & *\\
. & . & 5 & 5 & . & . & * & * & . & . & * & * & . & . & * & *\\
. & . & 5 & 5 & . & . & * & * & . & . & * & * & . & . & * & *\\[1ex]
. & . & * & * & . & . & * & * & . & . & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & . & * & * & *\\
. & . & 5 & 5 & . & . & * & * & . & . & * & * & . & . & * & *\\
. & . & 5 & 5 & . & . & * & * & . & . & * & * & . & . & * & *
\end{array}\right]
\quad
\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 6.}}\\
* & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & * & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & . & . & * & *\\
. & . & . & * & . & . & * & * & . & . & * & * & . & . & * & *\\[1ex]
. & . & * & * & * & * & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & * & * & * & * & * & * & * & * & * & * & *\\
. & . & . & . & . & . & 6 & 6 & . & . & 6 & 6 & . & . & 6 & 6\\
. & . & . & . & . & . & 6 & 6 & . & . & 6 & 6 & . & . & 6 & 6\\[1ex]
. & . & * & * & . & . & * & * & * & * & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & * & * & * & * & * & * & *\\
. & . & . & . & . & . & 6 & 6 & . & . & * & * & . & . & * & *\\
. & . & . & . & . & . & 6 & 6 & . & . & * & * & . & . & * & *\\[1ex]
. & . & * & * & . & . & * & * & . & . & * & * & * & * & * & *\\
. & . & * & * & . & . & * & * & . & . & * & * & . & * & * & *\\
. & . & . & . & . & . & 6 & 6 & . & . & * & * & . & . & * & *\\
. & . & . & . & . & . & 6 & 6 & . & . & * & * & . & . & * & *
\end{array}\right]$$

$$\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 7.}}\\
\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&.&\ast&.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&.&.&.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&.&.&.&.&.&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&.&.&.&.&.&.&.&.&7&7&.&.&7&7\\
.&.&.&.&.&.&.&.&.&.&7&7&.&.&7&7\\
.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast&.&\ast&\ast&\ast\\
.&.&.&.&.&.&.&.&.&.&7&7&.&.&\ast&\ast\\
.&.&.&.&.&.&.&.&.&.&7&7&.&.&\ast&\ast
\end{array}\right]
\quad
\left[\begin{array}{cccccccccccccccc}
\multicolumn{16}{c}{\text{Step 8.}}\\
\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&.&\ast&.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&.&.&.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&.&.&.&.&.&\ast&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&\ast&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&.&.&.&.&.&.&.&.&\ast&\ast&.&.&\ast&\ast\\
.&.&.&.&.&.&.&.&.&.&.&\ast&.&.&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast&\ast&\ast&\ast&\ast\\
.&.&\ast&\ast&.&.&\ast&\ast&.&.&\ast&\ast&.&\ast&\ast&\ast\\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&8&8\\
.&.&.&.&.&.&.&.&.&.&.&.&.&.&8&8
\end{array}\right]$$

As can be seen from Example I, the result of the factorization is not two block triangular matrices, but *block-cyclic triangles.* A block-cyclic triangle can be permuted to a block triangular matrix, as discussed in §5.2. However, it is not necessary to carry out this permutation for the solution of the block-cyclic triangular system of equations. Indeed, it is desirable to use the block-cyclic triangle for the forward and back substitutions, since the substitution process is load balanced for the block-cyclic triangles. Using block triangular matrices stored in a square data array (A) allocated to nodes with a consecutive data allocation scheme would result in poor load balance. Before discussing block-cyclic triangular solvers, and the relationship between pivoting strategies and the block-cyclic elimination order, we consider rectangular nodal arrays.

5.1.2. Rectangular arrays of processing nodes. Rectangular nodal arrays result in different row and column orderings, since the length of the cycle for the cyclic elimination order is different for rows and columns. Let $\hat{P}_1$ give the relationship between consecutive order and block-cyclic order elimination for rows, and let $\hat{P}_2$ give the relationship between consecutive order and block-cyclic order elimination for columns. Then, $\hat{P}_1(i) = j$ means that the ith row to be eliminated is row j. Specifically, for a matrix allocated to the $p \times q$ nodal array with the consecutive (block) allocation scheme, as described in §5.1, the block-cyclic ordered elimination with a block size $b = 1$ yields

$$\hat{P}_1 = 1, \frac{N}{p} + 1, 2\frac{N}{p} + 1, \ldots, \qquad \hat{P}_2 = 1, \frac{N}{q} + 1, 2\frac{N}{q} + 1, \ldots.$$

Thus, $\hat{P}_1 \neq \hat{P}_2$ if $p \neq q$. Clearly, if the number of nodes in the entire machine is not a square (as is the case for a 2048-node Connection Machine system CM-200, and the Intel Delta [30]), and all nodes are used, then $p \neq q$ and $\hat{P}_1 \neq \hat{P}_2$. How to choose p and q for optimum performance under the constraint $pq = \text{const}$ is discussed in §7.

Note that even though $\hat{P}_1 \neq \hat{P}_2$ naturally occurs when $p \neq q$, choosing a different blocking factor for rows and columns yields the same result.

5.2. Solving block-cyclic triangular systems. In an algorithm eliminating rows and columns consecutively, transformations are applied to an original matrix A, until it has been reduced to an upper triangular matrix T. In block-cyclic-ordered elimination, A is reduced to a block-cyclic triangular matrix $V = P_1 T P_2^{-1}$, where T is upper triangular. Block-cyclic triangles are just as easy to invert as standard block triangles, but they are load balanced across the nodes of a distributed memory machine. The block-cyclic triangle V for the example above (Step 8, Example I), and the corresponding permutation matrix P_1, are shown below. Example II shows a block-cyclic triangle for a rectangular nodal array ($P_1 \neq P_2$).

Example I: $N = 16, p = q = 4, b = 2$.

$$V = P_1 T P_2^{-1} = \left[\begin{array}{cccc@{\quad}cccc@{\quad}cccc@{\quad}cccc}
\ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast \\
\cdot & \cdot & \cdot & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast \\[1ex]
\cdot & \cdot & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \cdot & \ast & \ast & \cdot & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast \\[1ex]
\cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \ast & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \ast & \cdot & \cdot & \ast & \ast \\[1ex]
\cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \ast & \ast & \ast & \ast \\
\cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \cdot & \ast & \ast & \cdot & \ast & \ast & \ast \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \ast & \ast \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \ast
\end{array}\right],$$

where

$$P_1 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

The permutation matrix P_2 is constructed analogously.

Example II: $N = 16, p = 2, q = 4, b = 2$.

$$V = P_1 T P_2^{-1} = \left[\begin{array}{cccc@{\quad}cccc@{\quad}cccc@{\quad}cccc}
* & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\
. & * & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\
. & . & * & * & . & . & * & * & * & * & * & * & * & * & * & * \\
. & . & * & * & . & . & * & * & . & * & * & * & * & * & * & * \\
. & . & * & * & . & . & * & * & . & . & * & * & . & . & * & * \\
. & . & . & * & . & . & * & * & . & . & * & * & . & . & * & * \\
. & . & . & . & . & . & . & . & . & . & * & * & . & . & * & * \\
. & . & . & . & . & . & . & . & . & . & . & * & . & . & * & * \\[1em]
. & . & * & * & * & * & * & * & * & * & * & * & * & * & * & * \\
. & . & * & * & . & * & * & * & * & * & * & * & * & * & * & * \\
. & . & * & * & . & . & * & * & . & . & * & * & * & * & * & * \\
. & . & * & * & . & . & * & * & . & . & * & * & . & * & * & * \\
. & . & . & . & . & . & * & * & . & . & * & * & . & . & * & * \\
. & . & . & . & . & . & . & * & . & . & * & * & . & . & * & * \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & * & * \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & *
\end{array}\right].$$

In our block-cyclic triangular solver, the solution matrix X overwrites the right-hand sides Y. The solver requires that the set of right-hand sides Y and the matrix A are *aligned*. The alignment of Y and A ensures that the shape of the nodal array is the same for A and Y. For a single right-hand side, the alignment implies that the components of Y are allocated to the first column of the nodal array using the consecutive allocation scheme. With several right-hand sides, the alignment implies that the consecutive allocation scheme is used to allocate right-hand sides and columns of the matrix A to the same number of node columns with each right-hand side being allocated to a single-node column. The alignment of A and Y can be accomplished without data motion by using the compiler directive ALIGN. Alignment at run-time generally will require data motion. It can be performed by the Connection Machine router. Our library routine validates that the arrays are aligned and, if the arrays are not prealigned, performs the alignment through a call to the router.

Our column-oriented algorithm for solving block-cyclic upper triangular systems of the form $P_1 T P_2^{-1} X = Y$ starts with the last block column of $P_1 T P_2^{-1}$ and progresses toward its first block column, in a way similar to a standard column-oriented upper triangular system solver. A multiple of each block column of $P_1 T P_2^{-1}$ is subtracted from Y in each backward elimination step. The multiple of block column k, which is subtracted from Y at the kth stage of this process, is the kth block component (row) of the solution vector X. If the kth block column consists of columns $\hat{P}_2(j)$, $\hat{P}_2(j+1), \ldots, \hat{P}_2(j+b-1)$, then the last b elements of each column in this block column are in rows $\hat{P}_1(j)$, $\hat{P}_1(j+1), \ldots, \hat{P}_1(j+b-1)$. If X overwrites Y, then it is natural to record the multiplier of the kth block column in components (rows) $\hat{P}_1(j), \hat{P}_1(j+1), \ldots, \hat{P}_1(j+b-1)$ of Y. But these components (rows) of the result belong in components (rows) $\hat{P}_2(j), \hat{P}_2(j+1), \ldots, \hat{P}_2(j+b-1)$ of Y. Hence, after completing the iterative process of block column subtractions, the permutation $P_2 P_1^{-1}$ (a generalized shuffle [19]) must be applied to Y to obtain X.

The ordering of Y after the in-place substitution process, and the correct ordering of the solution are illustrated in Fig. 2 for an 8×8 matrix mapped to a 2×4 nodal array. The numbers indicate the pivoting order for the factorization and the placement of the solution components after the in-place back substitution and after the postpermutation. In the example, the destination address in the postpermutation is obtained by a right cyclic shift of the row index in the solution matrix. For instance, with indices labeled from 0, the content in location 1 of Y is sent to location 4, the content of location 4 to location 2, and the content of location 2 to location 1.

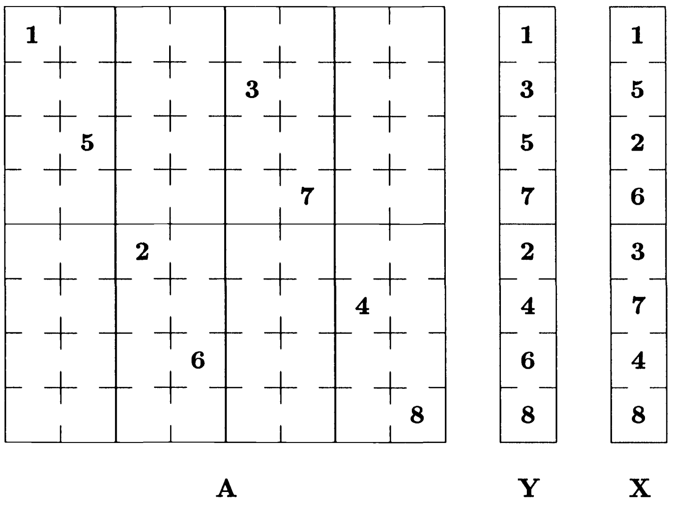


FIG. 2. *Block-cyclic triangular backsubstitution. In-place ordering* (Y) *and correct ordering* (X) *of solutions.*

Briefly put, the natural process for solving block-cyclic triangular systems evaluates $W = (P_1TP_1^{-1})^{-1}Y$. But $X = (P_1TP_2^{-1})^{-1}Y = (P_2P_1^{-1})(P_1TP_1^{-1})^{-1}Y = (P_2P_1^{-1})W$. Thus the solutions X, aligned with A and overwriting the aligned matrix Y, are obtained through a postpermutation $P_2P_1^{-1}$ applied to the result of the in-place block-cyclic triangular solver.

Note that in applying a block-cyclic elimination order to data allocated by a consecutive allocation scheme, the computations are load balanced both for factorization and triangular system solution without permuting the matrix A or the block-cyclic triangles. For instance, whenever $P_1 \neq P_2$ for rectangular nodal arrays, a shuffle permutation of the solution matrix is required. But, for few right-hand sides, the amount of data permuted is considerably less than if the input matrix and the right-hand side matrix had been permuted from consecutive to cyclic allocation and the solution matrix permuted from cyclic to consecutive allocation.

5.3. LU factorization and block-cyclic triangular system solution with partial pivoting. We now have presented all the elements of our algorithm for solving dense systems of equations with partial pivoting for a consecutive data allocation. The complete algorithm is as follows.

- Factor the matrix A in place using a block-cyclic elimination order. Exchange each selected pivot row with the appropriate row defined by the block-cyclic elimination order. Record the pivot selection.
- Align the right-hand side matrix Y with the matrix A (the array storing the block-cyclic triangles). Y and A can be aligned at compile time through the use of the compiler directive ALIGN.
- Perform a forward block-cyclic triangular system solve in place using the recorded pivoting information.
- Perform a backward block-cyclic triangular system solve in place.

• Whenever the row and column permutation matrices P_1 and P_2 are different, as in the case of rectangular nodal arrays, perform a postpermutation (generalized shuffle) of the solution matrix (or prepermutation (generalized shuffle) of the matrix A as explained in the next subsection).

• Align the solutions with Y, assuming that the solutions overwrite the right-hand sides.

No permutation of the matrix A or the block-cyclic triangular factors, is made for load balance. A permutation of the solution matrix is only required when $P_1 \neq P_2$, as in the case of rectangular nodal arrays. For a few right-hand sides, the size of the solution matrix is insignificant compared to the matrix A, and the time for permutation of the result is of little influence on the performance. For a large number of right-hand sides, a prepermutation of the matrix A, as explained in the next section, may yield better performance. In our implementation, the permutation of the result, when required, is performed by the Connection Machine system router. Optimal algorithms [19] are known for the generalized shuffle permutations required in pre or postpermutation when $N/\max(p,q)$ is a multiple of the block size. However, no implementation of such algorithms is currently available on the Connection Machine systems.

Note that if the data arrays A and Y are not aligned at compile time, then the last two steps could be combined into a single routing operation. Note further that if there are more right-hand sides than columns of A, then it may be preferable to align A with Y, if the optimal nodal array shapes for factorization and solution phases are different, or if alignment is made at run time, since fewer data elements must be moved if the layout of Y is the reference. Optimal array shapes are discussed further in §7.

In our implementation, the location of the pivot block row is defined by the block-cyclic elimination order. Selected pivot rows are exchanged with rows of the pivot block row in order to assure load balance.

The forward and backward solution in our implementation corresponds to the routine _TRSM in the level-3 BLAS, but it is not identical since a block-cyclic ordered elimination is used.

5.4. LU factorization and triangular system solution without pivoting: prepermutation vs. postpermutation. In §5.2 we showed that in case $P_1 \neq P_2$, backward elimination of a block-cyclic triangular system consists of two parts: a fairly standard backward elimination with reverse block-cyclic ordered block column subtractions from the right-hand side and a postpermutation of the result when $P_1 \neq P_2$. For such block-cyclic elimination orders, successive blocks are *not* selected from the diagonal of the matrix.

LU factorization without pivoting selects pivots from the diagonal of a matrix and works well for diagonally dominant systems precisely because the large diagonal elements are guaranteed to be the pivots in the factorization process. However, in block-cyclic ordered elimination with $\hat{P}_1 \neq \hat{P}_2$, the matrix entries used as pivots will be on the block-cyclic diagonal, i.e., in locations of the form $(\hat{P}_1(i), \hat{P}_2(i))$. Thus for diagonally dominant matrices factored on, for instance, a rectangular nodal array with a no-pivoting strategy, it is necessary to prepermute the original matrix in some way that maps the original diagonal to the block-cyclic diagonal.

One obvious prepermutation is to replace A with $B = P_1AP_2^{-1}$. Then, given a factorization $L_B^{-1}B = P_1T_BP_2^{-1}$ and routines to apply L_B^{-1} and $(P_1T_BP_1^{-1})^{-1}$, $AX = Y$ can be solved as $X = P_1^{-1}\ (P_1T_BP_1^{-1})^{-1}L_B^{-1}P_1Y$. But a more efficient approach is to replace A with $C = AP_1P_2^{-1}$. A block-cyclic elimination order applied to C yields $L_C^{-1}C = P_1T_CP_2^{-1}$. Furthermore, $L_C^{-1}A = P_1T_CP_1^{-1}$, and the solution of $AX = Y$ is obtained as $X = (P_1 T_C P_1^{-1})^{-1} L_C^{-1} Y$. In summary, a prepermutation of A required for numerical stability can be used to cancel the postpermutation of the result.

Thus in a block-cyclic elimination order on rectangular nodal arrays, LU factorization with pivoting on the diagonal is performed as follows:

Apply $P_1 P_2^{-1}$ from the right.	$A \leftarrow A P_1 P_2^{-1}$
Factor C in place.	$L_C, P_1 T_C P_2^{-1}$
Forward solve in place.	$Y \leftarrow L_C^{-1} Y$
Backward solve in place.	$X = (P_1 T_C P_1^{-1})^{-1} Y$

Note that if $P_1 = P_2$, then the first step has no effect on the data ordering and should be omitted. In this case, the factorization with no pivoting should be faster than with pivoting, since no time is required for finding pivots and for exchanging selected pivot rows with the rows of the block pivot row. However, when $P_1 \neq P_2$, then the no-pivot option requires a prepermutation of the matrix A, while the pivot option requires postpermutation of the solution matrix X in addition to the data rearrangement required for pivoting. In our implementation, we find that no pivoting is always faster than pivoting even when $P_1 \neq P_2$ and there is only one right-hand side (see Fig. 3).

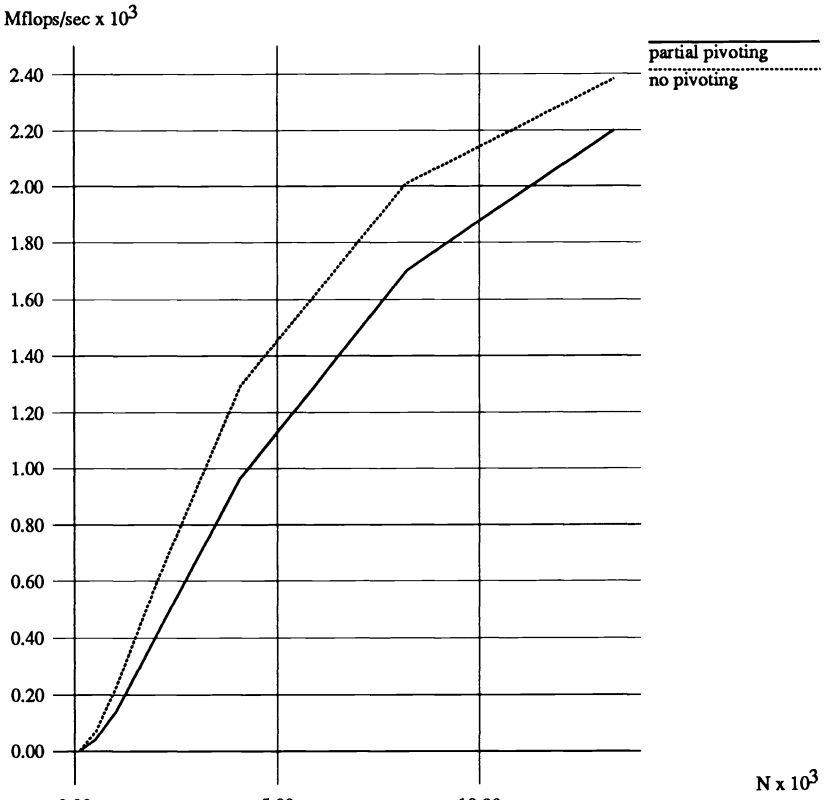


FIG. 3. *Execution rates for* LU *factorization with and without pivoting on a* 512*-node Connection Machine system* CM-200 *with blocking factor* 8, 64*-bit precision. Nodes in Gray code order.*

Note further that the prepermutation $A \leftarrow AP_1P_2^{-1}$ can be used also for the LU factorization with pivoting, thus removing the need for the postpermutation of the result. This technique gives improved performance when there are more columns in the right-hand side array Y than in the original matrix A. Thus prepermuting the matrix A when there are many right-hand sides should result in better performance for the no-pivoting option than for the pivoting option.

5.5. QR factorization and system solution. QR factorization and the solution of the factored matrix equations can be performed in a manner analogous to the LU factorization and the solution of triangular systems. In the factorization, pivoting is replaced by an inner product of the current column with all other columns, i.e., a vector-matrix multiplication. This vector-matrix multiplication doubles the number of floating-point operations and replaces two copy-spread operations with one physical spread-with-add operation. The result is an operator Q^T in factored form, satisfying $Q^TA = R$, where $R = P_1TP_2^{-1}$ is a block-cyclic upper triangular. If A has m rows and n columns with $m \geq n$, then the QR factorization may be used to find the vector X that minimizes the residual 2-norm $\|AX - Y\|_2$. The procedure is

$$\begin{array}{ll} \textit{Apply } Q^T. & Y \leftarrow Q^TY, \\ \textit{Backward solve.} & X = \Pi(P_2P_1^{-1})(P_1TP_1^{-1})^{-1}Y, \end{array}$$

where Π is projection onto the first n components. The backsolve $(P_1TP_1^{-1})^{-1}$ operates on the first n rows in block-cyclic order of Y. The final permutation $P_2P_1^{-1}$ moves the first n block-cyclic rows to the first n rows.

6. Performance. In this section we report first the performance achieved on a few matrix sizes. Then we analyze the performance and discuss factors that contribute to the discrepancy between the performance of the local matrix kernels and the performance of the global factor and solve routines. Communication and arithmetic are considered separately. All arrays are allocated with default compiler layouts. This implies that nodes are in Gray code order along each axis, and that Y is aligned with A only in the case that the number of right-hand sides is equal to the number of equations. A 2048-node and a 512-node Connection Machine system CM-200 were used for the performance measurements. Both systems result in rectangular nodal arrays. Alignment of Y with A (when necessary) and postpermutation of the solution(s) were performed using the Connection Machine router. Times for these data motion steps are included in the solve timings. In all cases we used random data for the matrix A. In this section, LU factorization always means LU factorization with partial pivoting. In this section, QR factorization always means QR factorization with no pivoting.

6.1. Measurements. In 64-bit precision the peak performance for LU factorization is 9.4 Gflops/s, or close to 4.6 Mflops/s per node, which is about 50% of the peak performance of the level-2 LBLAS. Table 6 shows the performance of LU and QR factorization for a few matrix sizes on a 2048-node Connection Machine system CM-200. The execution rate for LU factorization is computed based on $(2/3)N^3$ floating-point operations, and the QR factorization rate based on $\frac{4}{3}N^3$ operations, regardless of the blocking factor. For the solvers, the execution rates are based on $2RN^2$ operations for the LU solver and $3RN^2$ for the QR solver.

Tables 7 and 8 give some performance data for the factorization and the forward and backward solution of dense systems. R denotes the number of right-hand sides in the systems of equations being solved. The improved efficiency with an increasing matrix

TABLE 6

Execution rates in Mflops/s *for* LU *and* QR *factorization on a* 2048-*node Connection Machine system* CM-200. *A blocking factor b of* 16 *was used for* LU *factorization. A blocking factor of* 8 *was used for* QR *factorization. Nodes in Gray code order.*

No. of rows/ columns, N	LU		QR	
	32-bit	64-bit	32-bit	64-bit
1024	172	139	374	321
2048	620	494	1301	1053
4096	2040	1537	3902	2842
8192	5586	3846	8878	5704
16384	11573	7173	—	—
26624	16030	9398	—	—

size, increased number of right-hand sides, and increasing blocking factor is apparent. For LU factorization in 32-bit precision, the performance increases by a factor of over 700 for an increase in matrix dimension N by a factor of 132 and a blocking factor of 1. With a blocking factor of 16, the same increase in matrix size yields a performance increase by a factor of over 1300. The performance increase in 64-bit precision for an increase in N by a factor of 104 is 425 and 840, respectively. For small values of N, the performance increases much faster than N. For large values of N, the performance increase is still substantial and roughly proportional to N. This characteristic is true for both the factorization and the solution phases.

The execution rate for the forward and backward block-cyclic triangular system solvers increases significantly with the matrix size for a large number of right-hand sides. With the number of right-hand sides equal to the number of equations, the performance of the triangular solvers is about 50% higher than for LU factorization.

Comparing the execution rates of the LU and QR factorization routines, we observe that except for large matrices, the execution rates of the QR factorization routine are approximately twice that of the LU factorization routine. Hence, the execution times are approximately equal. However, for large matrices in 64-bit precision, the execution rate for QR factorization is only 15–20% higher than for LU decomposition, and the execution time for QR factorization is significantly longer than for LU factorization.

Comparing the LU and QR solve routines, we observe that the QR solve routine benefits more from an increased blocking factor than does the LU solve routine. The execution rate of the QR solve routine is about 20% higher than the LU solve routine for a large number of right-hand sides and a blocking factor of 1. But, for a blocking factor of 16, the QR solve routine has an execution rate that is about 40% higher than that of the LU solve routine.

The optimum blocking factors for LU factorization and triangular system solution are summarized in Fig. 4. The optimum blocking factor increases with an increased matrix size. For LU factorization the sensitivity of the execution rate with respect to the blocking factor is small (on the order of 20%) for small matrices, while for large matrices an increase in the blocking factor from 1 to 16 may increase the performance by a factor of more than two. The behavior is similar for the LU solve routine. The optimum blocking factor increases with the matrix size. In our implementation the optimum blocking factor for LU factorization is 4 for matrices of size up to 1024, 8 for matrices of size between 1024 and 8096, and 16 for sizes 8096 to 16896, the maximum size used in our test. Note that the optimum blocking factor for the LU solve routine is generally higher than for the factorization routine. Figures 5 and 6 show the execution rate as a

TABLE 7

Execution rates in Mflops/s *for* LU *factorization and block-cyclic triangular system solution on a* 512*-node Connection Machine system* CM-200 *as a function of matrix size, blocking factor, and the number of right-hand sides. Nodes in Gray code order.*

Matrix size	Block size	32-bit precision				64-bit precision			
		Factor	Solve			Factor	Solve		
N	b		$R=1$	$R=N/2$	$R=N$		$R=1$	$R=N/2$	$R=N$
128	1	2.7	.1	3.1	6.1	2.6	.1	3.1	6.0
512	1	39.9	.3	45.1	82.8	37.5	.2	42.0	72.8
1024	1	140.7	.5	153.0	252.0	126.4	.4	131.5	198.7
2048	1	420.1	.9	417.9	624.8	342.1	.7	320.7	434.8
4096	1	944.3	1.5	915.9	1127.9	667.7	1.1	615.1	713.1
8192	1	1510.1	2.2			964.7	1.5		
13312	1	1804.3	2.7			1104.8	1.8		
16896	1	1910.2	3.0						
128	4	3.3	.1	4.2	8.3	3.0	.1	4.1	8.0
512	4	48.8	.5	62.2	112.8	43.3	.4	55.9	96.7
1024	4	170.0	.9	213.1	348.8	146.7	.8	179.1	277.2
2048	4	517.8	1.7	614.9	892.7	420.3	1.5	470.8	639.3
4096	4	1237.5	3.1	1407.0	1726.8	920.0	2.6	970.9	1143.3
8192	4	2204.8	5.1			1506.6	3.9		
13312	4	2835.7	6.9			1857.9	4.8		
16896	4	3099.1	7.7						
128	8	3.2	.1	4.1	8.2	2.9	.1	3.9	7.6
512	8	48.6	.5	65.4	117.8	42.0	.5	57.4	98.8
1024	8	169.1	1.0	224.3	369.2	141.5	.9	184.4	286.5
2048	8	524.1	2.0	667.5	972.4	416.8	1.7	501.1	680.4
4096	8	1330.9	3.8	1603.3	2013.3	966.9	3.2	1075.8	1289.2
8192	8	2561.9	6.6			1701.1	5.1		
13312	8	3471.8	9.2			2200.0	6.7		
16896	8	3876.7	10.5						
128	16	3.1	.1	3.9	7.6	2.7	.1	3.4	6.7
512	16	43.8	.6	64.6	116.9	36.4	.5	55.2	95.6
1024	16	154.8	1.1	223.4	371.8	125.1	1.0	179.9	281.6
2048	16	485.5	2.2	680.9	979.2	374.6	1.9	498.1	674.0
4096	16	1268.3	4.2	1651.2	2085.3	907.1	3.5	1091.7	1317.4
8192	16	2574.8	7.5			1691.8	6.0		
13312	16	3628.9	10.8			2270.6	8.1		
16896	16	4116.9	12.6						

function of matrix size and blocking factor for LU factorization and solve, respectively. Figure 7 shows the execution rate of the LU solve routine as a function of matrix size and the number of right-hand sides.

The optimum blocking factor for the QR factorization routine increases somewhat slower with the matrix size than for the LU factorization routine. The optimum blocking factor for the QR solve routine is higher than for the QR factorization routine, just as in the case of LU factorization and solve. Figure 8 shows the execution rate as a function of matrix size and blocking factor for QR factorization.

In the next few subsections we analyze the performance behavior of the communication functions and the local arithmetic as a function of matrix and machine size and provide a model to predict the performance.

6.2. Communication. Two main types of communication are required: (i) *Spreads* copy data from a single row (or column) of nodes to all other rows (or columns); (ii) *Reductions* to select the pivot row in LU factorization with partial pivoting and for col-

TABLE 8

Execution rates in Mflops/s *for* QR *factorization and block-cyclic triangular system solution on a* 512-*node Connection Machine system* CM-200 *as a function of matrix size, blocking factor, and the number of right-hand sides. Nodes in Gray code order.*

Matrix size N	Block size b	32-bit precision				64-bit precision			
		Factor	Solve			Factor	Solve		
			$R=1$	$R=N/2$	$R=N$		$R=1$	$R=N/2$	$R=N$
128	1	5.8	.1	6.2	11.9	5.3	.1	5.6	10.5
512	1	85.7	.5	83.3	142.0	74.2	.5	66.9	104.6
1024	1	299.3	1.0	261.1	390.7	237.3	.8	186.9	255.6
2048	1	865.1	1.8	639.8	876.5	605.3	1.3	412.0	514.4
4096	1	1827.3	2.8	1297.0	1510.4	1104.9	1.9	744.7	827.1
8192	1	2750.0	4.0			1516.7	2.4		
13312	1	3183.7	4.8			1701.0	2.7		
16896	1	3334.3	5.1						
128	4	7.2	.4	9.8	19.1	6.3	.3	8.4	16.2
512	4	106.6	1.6	134.6	235.5	88.0	1.2	108.0	179.6
1024	4	371.5	3.0	436.0	675.1	284.7	2.2	320.4	469.6
2048	4	1101.4	5.5	1126.1	1534.4	749.3	3.9	748.3	960.1
4096	4	2427.1	9.1	2235.1	2601.2	1471.4	5.8	1311.3	1476.8
8192	4	3902.2	12.9			2178.2	7.6		
13312	4	4694.3	15.3			2544.4	8.6		
16896	4	4986.5	16.3						
128	8	7.0	.4	9.4	18.2	6.0	.3	7.8	14.8
512	8	102.9	2.3	146.2	253.9	80.6	1.6	111.6	184.5
1024	8	359.8	4.4	469.1	724.7	262.3	2.9	334.5	496.6
2048	8	1088.8	8.1	1272.7	1698.4	711.9	5.3	818.4	1034.9
4096	8	2548.5	13.8	2557.4	3012.5	1475.9	8.5	1555.8	1793.4
8192	8	4361.5	20.4			2315.9	11.5		
13312	8	5424.8	24.2			2797.1	13.2		
16896	8	5838.8	25.8						
128	16	6.3	.4	8.4	16.0	4.9	.3	6.3	11.7
512	16	85.6	2.8	143.2	251.7	63.4	1.8	105.5	176.6
1024	16	300.9	5.4	470.6	744.4	206.9	3.4	324.3	484.7
2048	16	917.4	10.2	1295.5	1729.7	572.9	6.2	814.3	1048.4
4096	16	2227.2	17.7	2762.9	3297.1	1253.9	10.2	1577.7	1827.3
8192	16	4068.2	26.9			2113.7	14.7		
13312	16	5276.3	32.5			2669.0	17.4		
16896	16	5780.5	34.8						

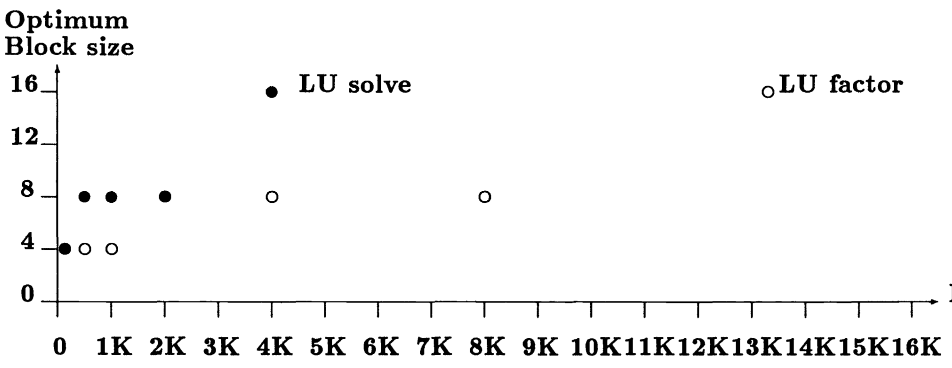


FIG. 4. *Optimal blocking factor for* LU *factorization and triangular system solution.*

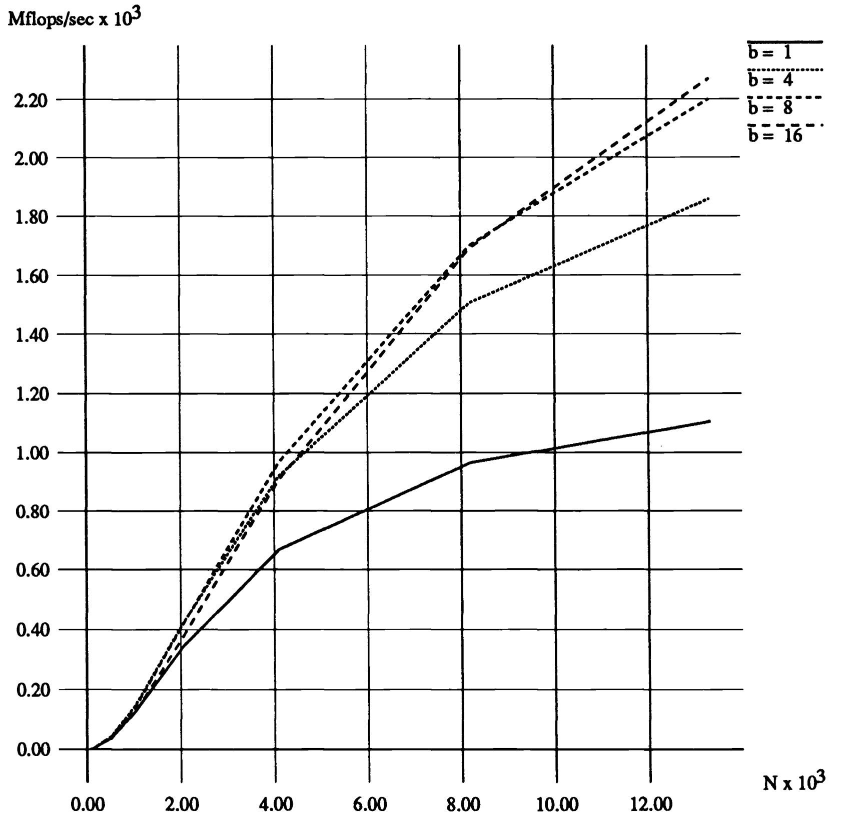


FIG. 5. *Execution rates for* LU *factorization on a* 512-*node Connection Machine system* CM-200 *with blocking factor b,* 64-*bit precision. Nodes in Gray code order.*

umn summations in QR factorization. In addition, general *sends* are used to align the right-hand sides Y with the columns of A, to realign the solutions with the input array Y, and, when $P_1 \neq P_2$, to prepermute the matrix A, or to postpermute the solution(s). In a spread of a pivot column, N/p elements per node must be communicated to every other node in a row for the consecutive data allocation described in §5.1. With a blocking factor of b rows and columns, the number of elements to be spread for a block column is Nb/p. Similarly, a block row requires that Nb/q elements be spread to every other node in a column.

On the Connection Machine system CM-200, where the nodes are interconnected as a Boolean cube, there exist several paths from the node that must spread the data to any of the nodes receiving the data. Indeed, since p and q are always chosen such that the nodes form subcubes of the Boolean cube, there exist $\log_2 p$ and $\log_2 q$ edge-disjoint paths between a node and every other node in a column and row subcube. Hence, the data set that a node must spread can be divided up among the different paths to balance the communications load and maximize the effective use of the available communications bandwidth [18]. Spreads on the Connection Machine system CM-200 are implemented in this manner and use the communications bandwidth optimally [18]. Note that for a fixed data set per node, the time for a spread decreases with an increased number of nodes. Table 9 gives some timings for spreads of different size data sets on Connection Machine systems CM-200 of various sizes. For a large data set per node, increasing

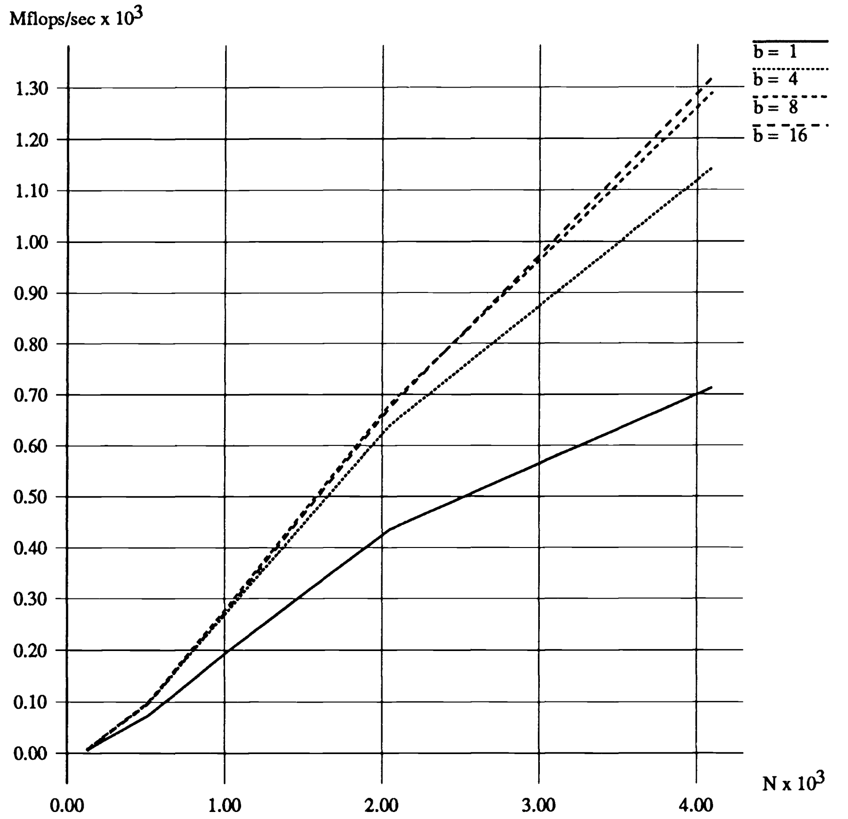


FIG. 6. *Execution rates for solution of block-cyclic triangular systems of equations with N right-hand sides for* LU *decomposition on a* 512*-node Connection Machine system* CM-200 *with a blocking factor b,* 64*-bit precision. Nodes in Gray code order.*

the number of cube dimensions by a factor of 10 (from 1 to 10) reduces the time for a spread by a factor of 3.41. Increasing the number of dimensions by a factor of 5 yields a reduction in the time for a spread of a large data set by a factor of 2.06. The overhead is quite substantial, making the speedup significantly less than the increase in the number of cube dimensions.

The prepermutation $P_1 P_2^{-1}$, or the postpermutation $P_2 P_1^{-1}$, when required, is performed by the Connection Machine system CM-200 router in our implementation. These permutations are *generalized shuffle* permutations. Optimal algorithms for Boolean cube networks are known for these permutations when $N/\max(p,q)$ is a multiple of the block size [19], but not implemented on the Connection Machine system CM-200. In cases with only one right-hand side, the cost of the alignment and postpermutation is only a small fraction of the total cost of the solve.

Note that, on the Connection Machine systems, the default layout of the matrix A and the right-hand sides Y is typically not the same, since the nodal array shape depends upon the data array shape. Aligning A and Y at compile time avoids data motion at run time. With a default layout of Y and A, the alignment constitutes a shuffle permutation, which would be performed by the router. Similarly, with the solutions overwriting the right-hand sides, the default data allocation requires a reallocation of the result from being aligned with A, to the default layout. This reallocation could be combined with

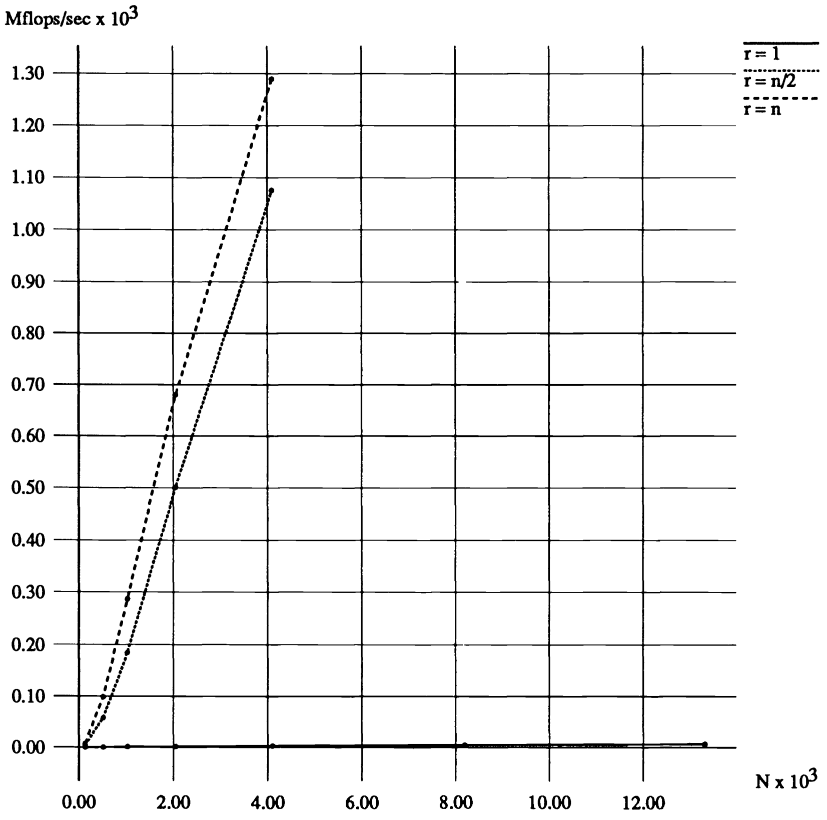


FIG. 7. *Execution rate for solution of block-cyclic triangular systems of equations with r right-hand sides for* LU *decomposition on a* 512*-node Connection Machine system* CM-200 *with a blocking factor* 8*,* 64*-bit precision. Nodes in Gray code order.*

TABLE 9
Time in msec *for spreads of different size data sets and Connection Machine systems* CM-200 *of various sizes:* 32*-bit precision.*

Number of elements	Number of nodes										
	2	4	8	16	32	64	128	256	512	1024	2048
2	0.0366	0.383	0.448	0.502	0.573	0.640	0.719	0.789	0.878	0.965	1.036
4	0.0366	0.383	0.448	0.502	0.573	0.640	0.719	0.789	0.878	0.965	1.036
8	0.0623	0.416	0.480	0.505	0.576	0.644	0.722	0.792	0.881	0.968	1.040
16	0.1136	0.470	0.515	0.542	0.615	0.684	0.763	0.798	0.888	0.975	1.046
32	0.2162	0.577	0.598	0.600	0.691	0.730	0.811	0.848	0.939	1.027	1.101
64	0.4214	0.793	0.741	0.720	0.773	0.856	0.907	0.947	1.042	1.133	1.168
128	0.8318	1.223	1.055	0.962	0.963	0.984	1.044	1.045	1.247	1.305	1.344
256	1.6527	2.084	1.654	1.445	1.377	1.352	1.360	1.338	1.410	1.434	1.503
512	3.3139	3.805	2.882	2.410	2.203	2.054	1.990	1.925	1.946	1.953	2.001
1024	6.7319	7.247	5.310	4.342	3.825	3.491	3.287	3.098	3.021	2.991	2.957
2048	13.4530	14.129	10.193	8.204	7.067	6.331	5.844	5.445	5.210	5.028	4.910
4096	26.9059	27.895	19.931	15.929	13.585	12.048	10.960	10.137	9.588	9.102	8.782
8192	53.8115	55.426	39.437	31.379	26.619	23.444	21.226	19.522	18.344	17.288	16.520
16384	107.6210	110.476	78.415	62.279	52.657	46.269	41.721	38.291	35.818	33.662	31.997
32768	215.2400	220.577	156.396	124.070	104.723	91.874	82.711	75.829	70.764	66.366	62.991

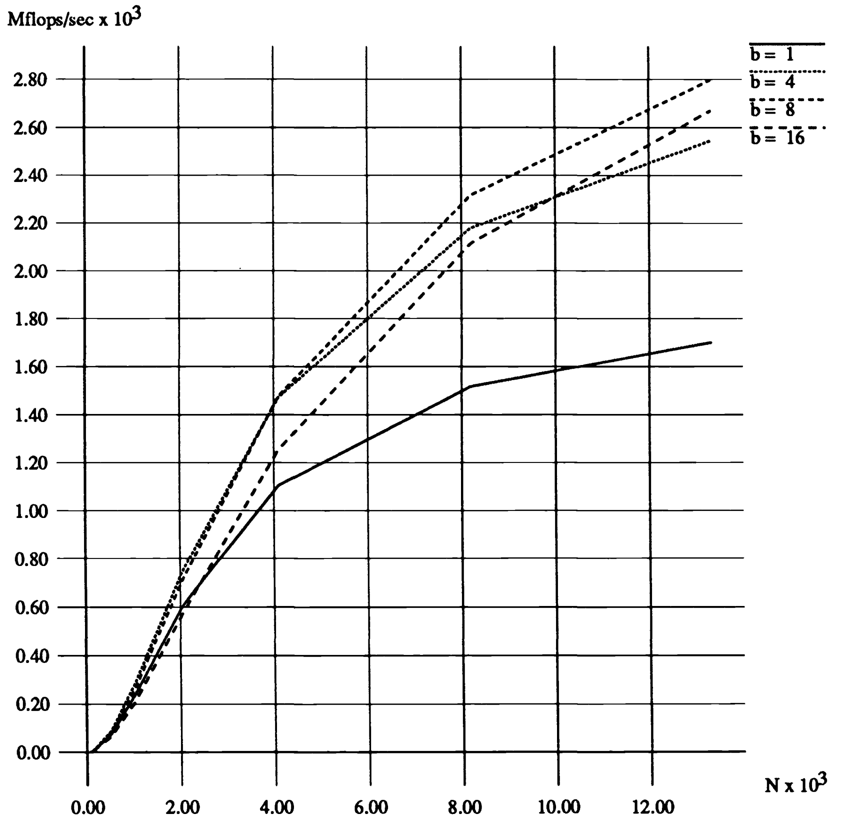


FIG. 8. *Execution rates for* QR *factorization on a* 2048*-node Connection Machine system* CM-200 *with a blocking factor b,* 64*-bit precision. Nodes in Gray code order.*

a postpermutation of the solution matrix X, when necessary, into a single permutation. Such a combined permutation would require the same time as either the postpermutation itself, or the reallocation of X, using optimal algorithms [19].

Note also, that if all arrays have a default layout, then for many right-hand sides it may be advantageous with respect to performance to align the matrix A with Y and, if a prepermutation of A is necessary, combine this permutation with the alignment operation.

6.3. Arithmetic efficiency. Not all the work is well load-balanced, even in the block-cyclic order elimination. Some work is applied only to a single row (or column) of the matrix at a time. For example, in LU factorization with partial pivoting, it is necessary to find the location of the largest element in the current column. Though this work is of order $O(N)$ for a single (block) row or column, compared to the $O(N^2)$ work for a rank-b update, it is magnified in importance because the grid-based layouts do not load balance individual rows and columns across the whole machine. Despite this drawback, grid-based layouts are optimal for communication systems in which the communication time is determined by the data volume leaving or entering a node, as shown in §7.

6.4. Detailed performance analysis. For simplicity, in this section and the next, we consider only the case $p = q$.

Assuming that αN^3 floating-point operations are carried out on p^2 nodes running at rate r, while the time per unit of work spent on N^2 work (some arithmetic and communication) on p nodes is C, the total time is $\alpha N^3/rp^2 + C(N^2/p)$. The achieved computational rate is

$$R = \frac{\alpha N^3}{\frac{\alpha N^3}{rp^2} + C\frac{N^2}{p}}$$

or

$$R = rp^2\left(\frac{\theta}{\theta+1}\right),$$

where

$$\theta = \left(\frac{N}{p}\right)\left(\frac{\alpha}{rC}\right).$$

For LU factorization on a 2048-node Connection Machine system CM-200, the compilers by default configure the nodes as a 32 x 64 array. The algorithm we use requires three spreads for each row and column that is eliminated; two to accomplish the pivoting row swap and one to spread the column of coefficients. From Table 9, we conclude that the time for a row spread is $570 + 3.1*$ (number of 32-bit words per column of nodes) msec, and the time for a column spread is $640 + 2.7*$ (number of 32-bit words per row of nodes) msec. For the largest matrices in Table 7, $N/p \approx 630$ (actually the submatrices assigned to each node are rectangular 448×896, and $\sqrt{(448 * 896)} \approx 630$). Some straightforward calculus shows that communication contributes about 15 msec to C. By counting cycles for the $O(N^2)$ arithmetic part of the code, we estimate that it contributes another 12 msec to C. For LU factorization $\alpha = 2/3$. We used a blocking factor of $b = 16$ for the timings reported in Table 7, i.e., all $O(N^3)$ work is performed by rank-16 updates. These run at a peak rate of about 7.6 Mflops/s per node. However, including the effect of DRAM page faults reduces this performance to a little over 7 Mflops/s. Hence, $\alpha/(rC) \approx 2/(3\cdot 7.0\cdot 27) \approx 1/(285)$, and $\theta \approx 2.2$, for which the performance loss factor is $\theta/(\theta+1) \approx 0.69$. Altogether, this analysis predicts a performance of about 4.85 Mflops/s per node, which is very close to the measured performance.

The important conclusion is that for $\theta \ll 1$, or N/p small relative to rC/α, the performance increases roughly linearly in N/p. Because rC is large, nearly all interesting cases fall into this range of submatrix sizes.

A minor point relevant for fine tuning is that smaller problems should use smaller block sizes. This need arises because one of the contributions to C is work occurring entirely within block rows or block columns, which grows quadratically with the block size b.

6.5. Scalability. As the matrix size N increases for a fixed machine size, the number of rows and columns assigned to a node increases proportionally. Hence, if N is doubled, so is the amount of data per row *and* column per node. Hence, the time required for the $O(N^2)$ term increases in proportion to N^2. But the amount of work that must be performed by each node for the factorization increases in proportion to the cube of the local matrix size. Hence, the importance of the $O(N^2)$ term decreases with increases N for a fixed number of nodes. However, the significance does not decrease in proportion to N, since the efficiency of the level-2 LBLAS increases with increased N. The variation in the computational rate r spans a range of more than one order of magnitude, while the performance loss factor may span two orders of magnitude. Hence, the dramatic variation in performance as a function of the local submatrix size.

Scaling the problem size with the number of nodes such that size of the submatrix assigned to each node remains fixed increases the efficiency of our current Connection Machine system CM-200 implementation. The performance of the level-2 LBLAS is unaffected by this form of scaling, but the communication is sped up because more channels are available for spreads and reductions. Thus, increasing the number of nodes decreases the value of C.

7. Optimal layouts. Communication is inevitable for the solution of full rank factorization problems on a distributed memory computer [9]. Another source of inefficiency in our implementation is the poorly load-balanced $O(N^2)$ work. We now consider the impact on the data communication (for spreads and reductions) of some alternative data layouts intended to improve the load balance for the $O(N^2)$ work.

Consider an arbitrary layout of an $N \times N$ matrix on p^2 nodes, which is regular in the following sense:

Each column intersects exactly c nodes in N/c elements.
Each row intersects exactly r nodes in N/r elements.
There are exactly N^2/p^2 elements on each node.
Each node intersects exactly cN/p^2 columns.
Each node intersects exactly rN/p^2 rows.

Figure 9 shows a two-dimensional data array assigned to nodes forming a one-dimensional array in a Boolean cube. A binary-reflected Gray code encoding [25] is used for the array embedding. Each square subarray of the data array is assigned to one of four nodes, as indicated by the number in the corresponding box. In effect, the data array is assigned to the nodes of the embedded one-dimensional array in a block-skewed way. This layout satisfies the regularity conditions with $c = 4$, $r = 4$, and $p = 2$.

0	1	3	2
1	0	2	3
3	2	0	1
2	3	1	0

FIG. 9. *Layout of a two-dimensional data array on a Boolean cube configured as a one-dimensional array.*

It is clear that by making $rc > p^2$, the load balance can be improved for row-and-column oriented operations. To evaluate the impact on the time for a spread, we assume that the time of the spread is proportional to max(*inject, eject*), where *inject* is the maximum number of data elements injected into the communications system by any node, and *eject* is the maximum number of elements ejected from the communications system by any node. Hence, we assume that communication operations are limited by the bandwidth at the nodes. Then, the time of a column spread is $\max(N/c, rN/p^2)$ and the time of a row spread is $\max(N/r, cN/p^2)$. Now, consider an algorithm consisting of local computation, row spreads, and column spreads. Suppose that local computation consists of a perfectly load-balanced part, a part which takes place on a single column at one time, and a part which takes place on a single row at one time. By making $rc > p^2$, we can reduce the cost of the single-column and single-row operations. In return, the cost of the row spreads and the column spreads will grow. In fact, it is likely that distributing rows and columns across more processors will also increase the cost of the perfectly load-balanced part of the computation, since more distributed layouts will reduce local vector lengths. If we ignore this effect of reduced vector lengths, then an optimal layout should minimize $cost = \alpha * \max(N/c, rN/p^2) + \beta * \max(N/r, cN/p^2) + \gamma * N/r + \delta * N/c$, where α, β, γ, and δ, are parameters describing the relative importance of column spread, row spread, single-row work, and single-column work, respectively.

To find the values of c and r that minimize this expression, we first assume that $rc = K$ with $K \geq p^2$. Then, $cN/p^2 \geq N/r$ and $rN/p^2 \geq N/c$, and the minimization problem reduces to finding the minimum of $(\alpha/p^2) * r + (\beta/p^2) * c + \gamma/r + \delta/c$, subject to the constraint $rc = K$. Using the constraint to eliminate c, and solving for the critical value of r, gives

$$r^2 = \frac{(\beta/p^2) * K + \gamma}{(\alpha/p^2) + \delta/K}.$$

Here, $\text{cost}(K) = 2 * \sqrt{((\alpha/p^2) + \delta/K) * ((\beta/p^2) * K + \gamma)}$. Minimizing this minimum cost with respect to K gives $K_{\min} = \sqrt{(\gamma * \delta)/(\alpha * \beta)} * p^2$. For $K > K_{\min}$, $\text{cost}(K)$ is increasing. So if $\gamma\delta < \alpha\beta$, the minimum cost for all $K \geq p^2$ occurs for $K = p^2$. If we take $K < p^2$, then the minimization problem reduces to finding the minimum of $(\alpha + \delta)/c + (\beta + \gamma)/r$ subject to the constraint $rc = K$. In this case, the minimum is a decreasing function of K, so the minimum cost for all $K < p^2$ occurs for $K = p^2$.

The above analysis assumes that the number of columns is equal to the number of rows, which is true for LU factorization on a matrix that fits in primary storage. For the triangular solve, the number of right-hand sides R is often not equal to the number of rows N. With R right-hand sides, the assumption that each column intersects c nodes in N/c elements is still valid. But each row now intersects r nodes in R/r elements. With NR/p^2 elements on each node, each node intersects cR/p^2 columns (instead of cN/p^2 columns) and rN/p^2 rows, as before. It is easily verified that even for the triangular solve, $K = p^2$ is optimal.

The overall conclusion is that, provided $\gamma\delta < \alpha\beta$, i.e., provided the geometric mean of the single-row and single-column work is less than the geometric mean of the row and column spread work, the best choice is $K = p^2$. The condition $rc = p^2$ implies that the optimal layout is indeed based on two-dimensional subgrids for both factorization and triangular system solution. This conclusion is true, because any column intersects a node in N/c elements and $N/c = rN/p^2$, which is the number of rows that meet any node. Thus, the rows meeting any node must be exactly those N/c rows that meet any one of the columns meeting that node. In our implementations of LU and QR factorization, the condition $\gamma\delta < \alpha\beta$ is satisfied (although not by a wide margin). The optimal value of r/c under the constraint $rc = p^2$ is $\frac{\beta+\gamma}{\alpha+\delta}\ \frac{R}{N}$. Thus, the ratio of the lengths of the axes of the two-dimensional nodal array is proportional to the ratio between the corresponding axes of the data array operated upon for both factorization and triangular system solution.

The above communications model applies to some, but not all architectures. For instance, for hypercubes with concurrent communication on all channels of every node, such as the Connection Machine system CM-200, the above communications model must be modified. In the example given by the diagram above, a column spread and a row spread are actually all-to-all broadcasts [2], [18]. The time for such an operation is proportional to the number of elements entering a node divided by the number of communication channels per node, i.e., the number of hypercube dimensions spanning their set of nodes involved in the all-to-all broadcast. For a two-dimensional hypercube, all-to-all broadcast is no more expensive than a standard column spread. Although twice as much data must be received by each node, there are twice as many channels available for the data to use. Therefore, on a four-node CM-200, the data layout shown in Fig. 9 would give better performance for dense linear algebra than any layout currently used, because with the Fig. 9 layout and cyclic elimination order, all vector operations would be perfectly load balanced.

8. Summary. We describe LU and QR factorization and solve algorithms for a block-cyclic ordered elimination for both square and rectangular nodal arrays. We show how prepermutation can be performed to guarantee pivoting on the diagonal for diagonally dominant matrices, without a need for postpermutation. The algorithms have been implemented on the Connection Machine systems CM-2 and CM-200 and are part of the CMSSL [27]. The peak execution rate of the LU factorization routine on a Connection Machine system CM-200 is about 9.4 Gflops/s in 64-bit precision.

The routines accept any data layout that can be created in the higher-level languages on the Connection Machine systems, either by the default layout rules, or through the use of compiler directives. The routines also perform operations on multiple instances concurrently, with instances distributed over different sets of nodes. The algorithms use standard communication functions, such as multiple instance (segmented) broadcast and reduction, and generalized shuffle permutations. Optimized routines are used for broadcast and reduction, while the router currently is used for the generalized shuffle permutations.

For small matrices the *execution times* for QR factorization without pivoting and LU factorization with partial pivoting are comparable, while for large matrices the *execution rates* become comparable, and hence the execution time for LU factorization with partial pivoting is considerably shorter. The execution rate for the block cyclic triangular solve for LU decomposition is 50–100% higher than for the factorization for a number of right-hand sides equal to the number of unknowns. The behavior is similar for QR factorization.

The value of blocking operations on rows and columns increases with matrix size. For small matrices, blocking may yield an enhanced execution rate by 20%, while for large matrices the blocking may offer an increased execution rate of about a factor of two. In our implementations, the optimum blocking factor for LU factorization with partial pivoting increases from 4 to 16 with the matrix size increasing from 128 to 16896. The optimum blocking factor for QR decomposition without pivoting was observed to increase slower with the matrix size than for LU factorization. The optimum blocking factors for the solve routines were always higher than for the factorization routines.

The execution rate for LU factorization without partial pivoting is higher than when partial pivoting is used. The difference in performance depends upon the problem size, but is typically in the 10–20% range.

The peak performance for the global factorization routines is about two-thirds of the peak performance of the local level-2 BLAS routines used for the $O(N^3)$ work in the factorization (but approximately equal to the performance of the global matrix multiplication routine in the CMSSL). The $O(N^3)$ work is well load balanced and performed at high efficiency through blocking of row and column operations. The $O(N^2)$ work introduces a significant performance penalty, even for very large matrices, and a few thousand nodes. For submatrices of a size of about 600×600, about one-third of the time is spent on $O(N^2)$ work on the Connection Machine system CM-200 with 2048 nodes. About two-thirds of this overhead time is spent in communication with the remainder spent in poorly load-balanced arithmetic.

We also show that for architectures with a very simple communication performance model and for data layouts from a very large regular family, it is not possible to eliminate the overhead from poorly load-balanced arithmetic. Specifically, we show that in a communication model in which the time for a spread or reduction is determined only by the amount of data that either leaves or enters a node regardless of the configuration of the nodes, assignment of subarrays to nodes based on a two-dimensional nodal array is optimal both for the factorization and the triangular system solution. We also show that the ratio between the lengths of the axes of the two-dimensional nodal array is proportional to the ratio between the corresponding axes of the data array operated upon, i.e., the matrix subject to factorization, or the set of right-hand sides for triangular system solution. This proof does not apply directly to the CM-200, because for hypercubes the available bandwidth for a spread depends upon the node configuration, and the simple communication model does not fully capture the communications capabilities.

Acknowledgments. Many people at Thinking Machines Corp. provided support for the dense linear algebra work reported here. These include Mark Bromley, Tim Harris, Steve Heller, Robert L. Krawitz, Doug MacDonald, Kapil K. Mathur, Jill P. Mesirov, Luis F. Ortiz, and Leo Unger. Special thanks go to Hank Dardy at the Naval Research Laboratories, Washington, D.C., who made the Laboratories' Connection Machine system CM-200 available for the benchmarks presented in Tables 7 and 8. Robert Whaley's assistance in this effort is greatly appreciated. Last, but not least, we would like to thank the referees for their careful reading of the manuscript and several helpful comments and suggestions.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *Preliminary LAPACK User's Guide*, Tech. Rep., University of Tennessee, July 1991.

[2] J.-P. Brunet and S. L. Johnsson, *All-to-all broadcast with applications on the Connection Machine*, Internat. J. Supercomputer Appl., 3 (1992).

[3] M. Y. Chan, *Embedding of grids into optimal hypercubes*, SIAM J. Comput., 5 (1991), pp. 834–864.

[4] J. J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, Tech. Rep. Reprint No. 1, Argonne National Laboratories, Mathematics and Computer Science Division, Argonne, IL, August 1988.

[5] ———, *A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs*, Tech. Rep. Reprint No. 2, Argonne National Laboratories, Mathematics and Computer Science Division, Argonne, IL, August 1988.

[6] J. J. Dongarra, J. DuCroz, S. Hammarling, and R. J. Hanson, *An Extended Set of Fortran Basic Linear Algebra Subprograms*, Tech. Rep. Tech. Memo. 41, Argonne National Laboratories, Mathematics and Computer Science Division, Argonne, IL, November 1986.

[7] G. C. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu, *Fortran D language specification*, Tech. Rep. TR90-141, Department of Computer Science, Rice University, Houston, TX, December 1990.

[8] G. C. FOX, M. A. JOHNSSON, G. A. LYZENGA, S. W. OTTO, J. K. SALMON, AND W. FURMANSKI, *Solving Problems on Concurrent Processors*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[9] W. M. GENTLEMAN, *Some complexity results for matrix computations on parallel processors*, J. ACM, 1 (1978), pp. 112–115.

[10] P. HARTEN, H.-Y. LIN, H. RAJIC, AND D. WELLS, BLAS *and* FFT *performance on the* Intel i860 *microprocessor*, Tech. Rep., 1991.

[11] I. HAVEL AND J. MÓRAVEK, *B-valuations of graphs*, Czech. Math. J., 22 (1972), pp. 338–351.

[12] C.-T. HO AND S. L. JOHNSSON, *Embedding meshes in Boolean cubes by graph decomposition*, J. Parallel Distributed Computing, 4 (1990), pp. 325–339.

[13] S. L. JOHNSSON, *Band matrix systems solvers on ensemble architectures*, in Algorithms, Architecture, and the Future of Scientific Computation, University of Texas Press, Austin, TX, 1985, pp. 195–216.

[14] ———, *Dense matrix operations on a torus and a Boolean cube*, in The National Computer Conference, July 1985.

[15] ———, *Fast banded systems solvers for ensemble architectures*, Tech. Rep. YALEU/DCS/RR-379, Department of Computer Science, Yale University, New Haven, CT, March 1985.

[16] ———, *Solving narrow banded systems on ensemble architectures*, ACM Trans. on Math. Software, 3 (1985), pp. 271–288.

[17] ———, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distributed Computing, 2 (1987), pp. 133–172.

[18] S. L. JOHNSSON AND C.-T. HO, *Spanning graphs for optimum broadcasting and personalized communication in hypercubes*, IEEE Trans. Comput., 9 (1989), pp. 1249–1268.

[19] ———, *Generalized shuffle permutations on Boolean cubes*, J. Parallel Distributed Computing, 1 (1992), pp. 1–14.

[20] S. L. JOHNSSON AND L. F. ORTIZ, *Local Basic Linear Algebra Subroutines* (BLAS) *on the Connection Machine System* CM-200, Tech. Rep. TMC-226, Thinking Machines Corp., Cambridge, MA, April 1992.

[21] C. L. LAWSON, R. J. HANSON, D. R. KINCAID, AND F. T. KROGH, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Trans. on Math. Software, 3 (1979), pp. 308–323.

[22] G. LI AND T. F. COLEMAN, *A parallel triangular solver for a distributed memory multiprocessor*, SIAM J. Sci. Statist. Comput., 3 (1988), pp. 485–502.

[23] ———, *A new method for solving triangular systems on a distributed memory message-passing multiprocessor*, SIAM J. Sci. Statist. Comput., 2 (1989), pp. 382–396.

[24] K. K. MATHUR AND S. L. JOHNSSON, *Multiplication of Matrices of Arbitrary Shape on a Data Parallel Computer*, Tech. Rep. 216, Thinking Machines Corp., Cambridge, MA, December 1991.

[25] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[26] B. T. SMITH, J. M. BOYLE, J. J. DONGARRA, B. S. GARBOV, Y. IKEBE, V. C. KLEMA, AND C. B. MOLER, *Matrix Eigensystem Routines—EISENPACK Guide*, Lecture Notes in Computer Science, Vol. 6, Springer-Verlag, Berlin and New York, 1976.

[27] THINKING MACHINES CORP., CMSSL *for Fortran*, 1990.

[28] ———, CM-200 *Technical Summary*, 1991.

[29] ———, CM *Fortran optimization notes: slicewise model, version* 1.0, 1991.

[30] R. A. VAN DE GEIJN, *Massively parallel LINPACK benchmark on the Intel Touchstone Delta and* iPSC/860 *systems*, Tech. Rep., The University of Texas at Austin, July 1991.

[31] H. ZIMA, P. BREZANY, B. CHAPMAN, P. MEHROTRA, AND A. SCHWALD, *Vienna Fortran—A Language Specification version* 1.1, Tech. Rep., Institute for Computer Applications in Science and Engineering, Hampton, VA, Interim Rep. 21, March 1992.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1289–1299, November 1993

COMPUTING THE EXACT LEAST MEDIAN OF SQUARES ESTIMATE AND STABILITY DIAGNOSTICS IN MULTIPLE LINEAR REGRESSION*

ARNOLD J. STROMBERG†

Abstract. The difficulty in computing the least median of squares (LMS) estimate in multiple linear regression is due to the nondifferentiability and many local minima of the objective function. Several approximate, but not exact, algorithms have been suggested. This paper presents a method for computing the exact value of the LMS estimate in multiple linear regression. The LMS estimate is a special case of the least quantile of squares (LQS) estimate, which minimizes the qth smallest squared residual for a given data set. For LMS, $q = [n/2] + [(p+1)/2]$ where [ ] is the greatest integer function, n is the sample size, and p is the number of columns in the X matrix. The algorithm can compute a range of exact LQS estimates in multiple linear regression by considering $\binom{n}{p+1}$ possible θ values. It is based on the fact that each LQS estimate is the Chebyshev (or minimax) fit to some q element subset of the data. This yields a surprisingly easy algorithm for computing the exact LQS estimates. These and other estimates are used to study the stability of the LMS estimate in several examples.

Key words. least median of squares estimator, multiple linear regression, Chebyshev, minimax

AMS subject classifications. 62J05, 62J20, 65U05

1. Introduction.

Linear regression treats the problem of estimating θ_o where:

$$y_i = x_i\theta_o + \epsilon_i \quad i = 1, 2, \ldots, n,$$

where $(x_i, y_i) \in (R^p, R)$ are data points and θ_o is an unknown p-dimensional parameter vector and the ϵ_i are unknown errors. We will denote estimators of θ_o by $\hat{\theta}$. The residuals, $y_i - x_i\theta$, $i = 1, 2, \ldots, n$, are denoted $r_i(\theta)$. The best known estimators of θ_o is the least squares estimator $\hat{\theta}_{\text{LS}}$ which is

$$\underset{\theta}{\text{Argmin}} \quad \sum_{i=1}^{n} r_i^2(\theta).$$

The least squares estimator has the drawback that it is heavily influenced by outliers. It also suffers from the problem of masking [1]; that is, it is possible that multiple outliers may be present in the data set, yet they are not detected by common least squares diagnostic procedures.

The breakdown point of an estimator [5] has been shown to be a useful measure of the robustness of an estimator. It can be thought of as the least amount of arbitrary contamination that can drive the estimate to infinity. It is clear that the breakdown point of the least squares estimate in linear regression is $1/n$. Recent research [12], [13] has shown the usefulness of estimators with breakdown point approximately equal to $\frac{1}{2}$. These estimators ignore outliers and seem to be able to detect masking when least squares diagnostic procedures do not. The most studied, and probably most used, high breakdown estimator is the least median of squares (LMS) estimator [11]. It is denoted $\hat{\theta}_{\text{LMS}}$ and defined as

$$\underset{\theta}{\text{Argmin}} \quad \underset{1 \le i \le n}{\text{Median}} \; r_i^2(\theta).$$

*Received by the editors April 7, 1992; accepted for publication (in revised form) November 20, 1992.

†Department of Statistics, University of Kentucky, 817 Patterson Office Tower, Lexington, Kentucky 40506-0027. Partial support for this research was provided by National Science Foundation grant DMS-9204380 and National Security Agency grant MDA-904-92-H-3077.

To obtain the highest possible breakdown point for $\hat{\theta}_{\rm LMS}$ when the data are in general position, meaning that any p points give a unique determination of θ, the median is defined as the kth order statistic where $k = [n/2] + [(p+1)/2]$ and $[\cdot]$ indicates the greatest integer function. The LMS estimate is a special case of a more general estimate called the least quantile of squares (LQS) estimate [12, p. 124], which minimizes the qth smallest squared residual. The LMS estimate is used as a starting value for a number of other high breakdown estimators, including MM estimators [20], τ estimators [21], and the Simpson, Ruppert, and Carroll (SRC) estimator [16].

The stability of computed LMS estimates under minor shifts in the data is currently questioned by many statisticians. Hettmansperger and Sheather [9] and Ruppert [14] discuss the engine knock data where the PROGRESS approximation to the LMS estimate is vastly di erent for the original data and the data with one point changed slightly. In these articles, it is unclear whether this instability is due to the use of an approximation algorithm or is inherent in the LMS estimator. Their example is discussed in §3 of this paper. As discussed in [14], the problem with the LMS estimate is that it may be unstable for some data sets. This is undoubtedly due to its low efficiency and large number of local minima. Ruppert argues for the use of S estimators which have a 50% breakdown point, 28.7% asymptotic efficiency under normal errors, and apparently many fewer local minima than the LMS estimate. He suggests that extensive comparisons be done among S estimators, the LMS estimator, and other high breakdown estimators. The likely conclusion of such comparisons would be that S estimators are much more stable than LMS estimators, but it is also likely that the S estimate will be unstable for some data sets. For high breakdown estimates to be useful in practice, we must have methods for investigating the stability of the fit. In this paper we are concerned with whether or not the exact LMS estimate can be unstable for minor shifts in the data for some data sets, and if so, we are concerned with detecting instability of the LMS fit. Similar techniques could be used to detect instability in other high breakdown estimators.

Cook and Hawkins [3, p. 643] suggest that the LMS estimate be computed over a range of values for q (LQS) "to gain some reassurance that there is one consistent story in the data." Rousseeuw and van Zomeren agree [13, p. 649]. This paper provides a method for computing a range of LQS estimates exactly, and thus the exact LMS estimate is computed. With approximate algorithms, the "story" for di erent values of q may be di erent because of the algorithm used. The exact algorithm presented here rules out that possibility. The algorithm also provides an additional check on the stability of the LMS estimate by computing the exact LMS estimate for all $n-1$ element subsets of the data at the same time it is computing the exact LQS estimates for a range of values of q. Again, the goal here is to investigate the consistency of the story told by the high breakdown estimate.

One of the drawbacks of the LMS estimate and LQS estimates is that they are quite difficult to compute. The objective function is continuous, but not di erentiable, and it has many local minima. The PROGRESS algorithm of Rousseeuw and Leroy [12] is the most widely used algorithm for estimating $\hat{\theta}_{\rm LMS}$ in linear regression. It could easily be modified to compute LQS estimates. For a given data set and regression function, the PROGRESS algorithm computes the exact fit, $\hat{\theta}_{ef}$, to many randomly chosen p point elemental subsets of the data set. Denote the $\hat{\theta}_{ef}$ with the smallest median squared residual $\hat{\theta}$. If the regression function has no intercept, $\hat{\theta}$ is the PROGRESS estimate of $\hat{\theta}_{\rm LMS}$. If an intercept is used in the model, the intercept of $\hat{\theta}$ is adjusted to yield the smallest possible median residual. This adjusted $\hat{\theta}$ is then the PROGRESS estimate of $\hat{\theta}_{\rm LMS}$. Rousseeuw and Leroy [12] note that at the expense of additional computation time, the intercept adjustment can be done for each elemental set. Unfortunately the latter algorithm, which Steele and Steiger [17] show will find the exact value of $\hat{\theta}_{\mathrm{LMS}}$ when $p = 2$, does not yield the exact LMS estimate in multiple linear regression when $p > 2$.

The MVELMS algorithm of Hawkins and Simonoff [8], which is also based on the selection of p point elemental sets, but which uses an intercept adjustment for all elemental sets, has been proposed as an alternative to the PROGRESS algorithm. In general, it produces estimates of θ_o with a smaller objective function than the PROGRESS algorithm.

Using a geometric argument, Tichavsky [18] argued that the exact LMS estimate in multiple linear regression can be found by considering all $p+1$ point elemental sets and for each elemental set finding the values of θ where the magnitudes, but not the signs, of all $p+1$ residuals are equal. The method leads to $(2^p - 1)\binom{n}{p+1}$ values of θ that must be considered in order to compute the exact value of $\hat{\theta}_{\mathrm{LMS}}$ in multiple linear regression. Given the complexity of the problem for moderately large n and p, Tichavsky suggests approximating $\hat{\theta}_{\mathrm{LMS}}$ by selecting p point elemental sets and checking the median squared residual for the values of θ generated by the selected elemental sets.

Because an LQS estimate minimizes the qth smallest residual for a given data set, it must minimize the maximum squared residual for some q element subset of the data. Thus, the qth LQS estimate is the Chebyshev (or minimax) fit (see, e.g., Cheney [2]) to that q element subset. Section 2 presents two theorems that can be used to find the Chebyshev fit for given data set. The first implies that each LQS estimate must be the Chebyshev fit to some $p+1$ element subset of the data, and the second provides a surprisingly easy method for computing the Chebyshev fit to $p+1$ points. The theorems are used to develop an algorithm that, by considering $\binom{n}{p+1}$ possible θ values, computes exact LQS estimates in multiple linear regression. At the same time, the algorithm computes the LQS estimate for a range of values of q and for $n-1$ element subsets of the data as a check on the stability of the LMS estimate. Section 3 presents several examples.

2. The Chebyshev fit. In this section we adapt theorems found in Cheney [2] to provide a method for computing the Chebyshev fit, and thus the LQS estimates, in linear regression. The first relevant theorem can be restated in the context of regression as follows.

THEOREM 1 [2, p. 36]. *In linear regression, the Chebyshev fit, $\hat{\theta}_c$, will be the Chebyshev fit to some $p+1$ element subset of the data.*

By Theorem 1, if we can find $\hat{\theta}_c$ for all $p+1$ element subsets of the data, then we can find $\hat{\theta}_c$ for the entire data set. Theorem 2 provides a method for finding $\hat{\theta}_c$ when the sample size is $p+1$. The Haar condition says that there is one and only one exact fit to any p points. More formally, a set of points in $\mathbb{R}^p$ satisfies the Haar condition if every subset of p points is linearly independent (see, e.g., [2, p. 45].) Let $Y = (y_1, y_2, \ldots, y_n)^T$ and x be the $n \times p$ matrix given by $(x_1, x_2, \ldots, x_n)^T$.

THEOREM 2 [2, p. 41]. *Consider the linear regression setting described in* §1 *with sample size $p+1$. Assume that the Haar condition is satisfied. Then*

$$\hat{\theta}_{\mathrm{LS}} = \mathrm{MY}, \quad \textit{where } \mathrm{M} = (X^T X)^{-1} X^T$$

is the least squares fit to the data. Let

$$\epsilon = \frac{\sum_{i=1}^{p+1} r_i^2(\hat{\theta}_{\mathrm{LS}})}{\sum_{i=1}^{p+1} \left| r_i(\hat{\theta}_{\mathrm{LS}}) \right|}$$

and S *be the* $p+1$ *dimensional vector where* $s_i = \mathrm{sgn}(r_i(\hat{\theta}_{\mathrm{LS}}))$, $i = 1, 2, \ldots, p+1$. *Then*

$$\hat{\theta}_c = \mathrm{M}(\mathrm{Y} - \epsilon \mathrm{S}).$$

Remarks. Since the LQS estimate, denoted $\hat{\theta}_{(q)}$, is the Chebyshev fit for some sample of $p+1$ points, the following algorithm (Fig. 2.1) can be used for computing the exact value of the LQS estimates in multiple linear regression: for each $p+1$ point elemental set, use Theorem 2 to compute the Chebyshev fit, denoted $\hat{\theta}_c$. The $\hat{\theta}_c$ with the least qth smallest squared residual will be the exact value of $\hat{\theta}_{(q)}$. As with the algorithms of §1, implementations should take advantage of the fact that for many $\hat{\theta}_c$, computing all the squared residuals and/or the sort to find the qth smallest squared residual can be avoided. Suppose $\hat{\theta}$ is the current best estimate of $\hat{\theta}_{(q)}$ and $\hat{\theta}_c$ is the Chebyshev fit to the $p+1$ point elemental set being considered. The squared residuals at $\hat{\theta}_c$ need only be computed until $n-q$ are more than $r^2_{(q)}(\hat{\theta})$ because then it must be that $r^2_{(q)}(\hat{\theta}_c) > r^2_{(q)}(\hat{\theta})$. Should this not be the case, $\hat{\theta}_c$ becomes the new estimate of $\hat{\theta}_{(q)}$ and the squared residuals are sorted to find the qth smallest squared residual at the new estimate of $\hat{\theta}_{(q)}$.

The fact that $\hat{\theta}_{(q)}$ is the Chebyshev fit to some $p+1$ point elemental set seems intuitive, but it is quite surprising that the computation of $\hat{\theta}_c$ provided by Theorem 2 is only moderately more computationally difficult than computing the exact fit, $\hat{\theta}_{ef}$ to p points as is done in the approximate algorithms of §1. This suggests that algorithms based on computing $\hat{\theta}_{ef}$ could be improved by computing $\hat{\theta}_c$ instead of $\hat{\theta}_{ef}$. For example, the PROGRESS algorithm could be significantly improved by having it compute $\hat{\theta}_c$ for $p+1$ point subsets instead of $\hat{\theta}_{ef}$ for p point subsets. Because of the complexity of computing $\hat{\theta}_c$ for all $p+1$ point subsets of the data, the exact LMS fit can only be computed for reasonably small n and p, say $n \leq 50$ and $p \leq 5$. Using the GAUSS programming language on an IBM 386-33, such a computation could be expected to take about 33 hours. It seems clear that there is still a need for appropriately modified approximate algorithms. One such algorithm is discussed in Hawkins [7].

The theorems generalize results of Steel and Steiger [17]. If the Chebyshev fits are distinct, then the LMS fit will have $p+1$ points with squared residuals equal to the median squared residual, $q-p-1$ points with squared residuals less than the median squared residual, and $n-q$ points with squared residuals more than the median squared residual.

The exact algorithm can easily be modified to compute LQS estimates for a range of values of q in one pass through the $\hat{\theta}_c$. Use one row of a matrix Q to hold the current best estimate of $\hat{\theta}_{(q)}$ for the range of q's being computed. At each $\hat{\theta}_c$, compute and sort the squared residuals. Then update $\hat{\theta}_{(q)}$ if $r^2_{(q)}(\hat{\theta}_c)$ is less than the previous smallest value for $r^2_{(q)}(\theta)$. After considering all $\hat{\theta}_c$, the matrix K will contain the exact values of $\hat{\theta}_{(q)}$ for the values of q selected. As discussed in §1, the $\hat{\theta}_{(q)}$ can be used to check the stability of the LMS estimate.

The algorithm can also be modified to compute $\hat{\theta}_{-i}$, the LMS estimate for the data set with the ith data point deleted. This can be done at the same time as the computation

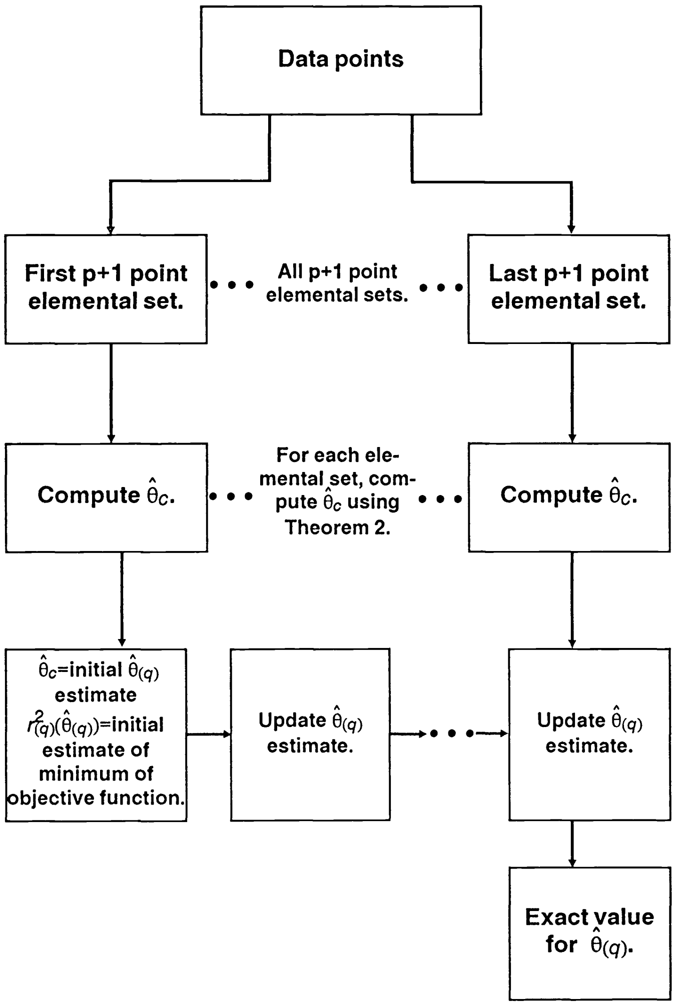

Data points
First p+1 point elemental set.
• • • All p+1 point elemental sets. • • •
Last p+1 point elemental set.
Compute θ̂c.
• • • For each elemental set, compute θ̂c using Theorem 2. • • •
Compute θ̂c.
θ̂c=initial θ̂(q) estimate r²(q)(θ̂(q))=initial estimate of minimum of objective function.
Update θ̂(q) estimate.
• • •
Update θ̂(q) estimate.
Exact value for θ̂(q).

FIG. 2.1. *Exact algorithm for computing* $\hat{\theta}_{(q)}$.

of the LQS estimates for the full data set. In general, use the ith row of an $n \times p$ matrix to hold the current best estimate of $\hat{\theta}_{-i}$. For each $\hat{\theta}_c$, check for improvement in each of the $\hat{\theta}_{-i}$. Of course, those $\hat{\theta}_c$ based on elemental sets that contain point i must be excluded from the possible $\hat{\theta}_{-i}$ values. For any other $\hat{\theta}_c$, the median squared residual for the data set with the ith point deleted will be the $(q + j)$th smallest residual for the entire data set where

$$j = \begin{cases} 0 & \text{if } r_i^2(\hat{\theta}_c) > r_{(q)}^2(\hat{\theta}_c), \\ 1 & \text{if } r_i^2(\hat{\theta}_c) \leq r_{(q)}^2(\hat{\theta}_c). \end{cases}$$

Thus the squared residuals need only be computed once at each $\hat{\theta}_c$ to find $\hat{\theta}_{(q)}$ for a range of q values and $\hat{\theta}_{-i}, i = 1, 2, \ldots, n$. As discussed in §1, these estimates can be used as a diagnostic tool. If their residuals are quite di erent than the exact LMS residuals for the entire data set, then the LMS estimate can be considered unstable.

The stability of the LMS fit could also be checked by listing fits to the N elemental sets with smallest values of the objective function. Instability in these fits would indicate instability in the LMS estimate.

The algorithms discussed here have been programmed by the author in GAUSS, version 2.0. A FORTRAN version is currently being developed. Its implementation will be considered elsewhere.

3. Examples.

3.1. Regression through the origin. The most notable difference between the approximate algorithms and the exact algorithm is that the elemental sets consist of $p + 1$ points for the exact algorithm, but only p points for the approximate algorithms. As an example of how this can affect the $\hat{\theta}_{\text{LMS}}$ fit, consider the data in Table 3.1, fit by a simple linear regression through the origin model. Both the PROGRESS and MVELMS algorithms use one point elemental sets, while the exact algorithm uses two point elemental sets. The PROGRESS and MVELMS algorithms find the line that passes through point 8 which has slope .657225 and median squared residual .107. According to this fit, points 0 through 4 should be considered outliers. The exact LMS fit is the Chebyshev fit to points 4 and 5, which has slope .38485 and median squared residual .075. According to the exact fit, points 6 through 9 should be considered outliers. The regression lines are depicted in Fig. 3.1.

The $\hat{\theta}_{-1}$ are useful in understanding the LMS fit to this data. If any of the first five points are removed, the LMS fit shifts to fit the upper five points, while removing any one of the upper five points has little impact on the LMS fit. It seems that considering any of the points to be outliers when the LMS fit to the $\hat{\theta}_{-1}$ is so variable is questionable.

TABLE 3.1
Data fit by simple linear regression through the origin.

Point #:	0	1	2	3	4	5	6	7	8	9
x:	1	2	3	4	5	1	2	3	4	5
y:	0.3302	0.6590	0.9888	1.3194	1.6495	0.6596	1.3192	1.9815	2.6289	3.3011

3.2. Cloud seeding data. This data, which can be found in Cook and Weisberg [4, p. 4], summarizes the results of a cloud seeding experiment done in Florida in 1975. On each of 24 days suitable for seeding, the following six explanatory variables were recorded:

A: "Action" was set to zero if no seeding took place and to one if seeding occurred.

B: "Time" was the number of days since the beginning of the experiment.

S: "Suitability" was a measure of the day's suitability for seeding.

C: "Echo coverage" was the percent of cloud coverage in the experimental area.

P: "Prewetness" was the total rainfall in the target area in the hour before seeding.

E: "Echo motion" was set to one for a moving radar echo and two for a stationary radar echo.

The data was fit using a multiple linear regression model with the preceding six explanatory variables and an intercept. The response variable was ln (rainfall) in a target area for a six-hour period. The PROGRESS approximation to $\hat{\theta}_{\text{LMS}}$ (1.43, .695, $-.016$, $-.455$, $-.039$, .941, 1.08) is based on the exact fit to points 0, 1, 9, 16, 17, 19, and 20. Its median squared residual was .0601. The MVELMS approximation to $\hat{\theta}_{\text{LMS}}$ (.740, 1.13,

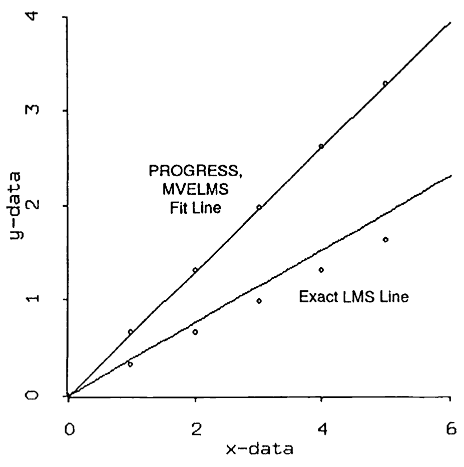


FIG. 3.1. PROGRESS, MVELMS, *and exact* LMS *fit of a simple linear regression through the origin model for the data in Table* 3.1.

$-.0047$, $-.567$, $-.056$, 3.60, .990) is based on the exact fit to points 2, 3, 5, 8, 11, 21, and 23. Its median squared residual was .0278. Neither of these approximations use a seven-point subset of the eight points (2, 3, 4, 8, 9, 11, 16, 23) that determine the exact LMS fit (.715, 1.13, $-.0052$, $-.551$, $-.056$, 3.61, .962) which has median squared residual .0241. The exact LMS computation took approximately 90 minutes on an IBM 386-33. The MVELMS basis does have five points in common with the exact basis, which explains why it is close to the exact LMS fit.

The plot of the LMS residuals versus fit values has been suggested [12] for assisting in detecting outliers in multiple linear regression. The PROGRESS, MVELMS, and exact LMS residual plots for the cloud seeding data are given in Fig. 3.2. Note that with the exception of point 6 and possibly 15, the PROGRESS algorithm identifies outliers di erent from the other two methods. The MVELMS plot is very close to the exact LMS plot, but there would be no way to know this without computing the exact LMS fit. For this data set, none of the residuals based on $\hat{\theta}_{-i}, i = 1, 2, \dots, n$ vary much from the residuals for the full data set (Fig. 3.2(a)); thus none of the data points are flagged as particularly influential.

The ability to compute the exact LMS estimate allows us to study the stability of $\hat{\theta}_{\text{LMS}}$ under shifts in the observed values. Let the modified cloud seeding data be the cloud seeding data with the response at point 4 shifted from 0.8961 to 1.1061. The residual plots for the three methods are almost identical to those given in Fig. 3.2. If we shift the response for point 4 from 1.1061 to 1.1161, the PROGRESS and MVELMS fits are virtually unchanged from those in Fig. 3.2, but the exact LMS residual plot is now similar to the PROGRESS plot of Fig. 3.2. It is interesting that, although the PROGRESS and exact LMS residual plots are similar in this case, the MVELMS fit has a smaller median squared residual than the PROGRESS fit. The instability of the LMS residual plot is shown in Fig. 3.3 where the shift of point 4 from 1.1061 to 1.1161 causes a di erent set of outliers to be identified. In the modified data set with point 4 at 1.1161, the influence of point 4 on the fit is evident from $\hat{\theta}_{-4}$, which yields a residual plot quite di erent from the residual plot for the entire data set (Fig. 3.3(b)). As expected, since only point 4 has been modified, the $\hat{\theta}_{-4}$ residual plot is similar to Fig. 3.3(a).

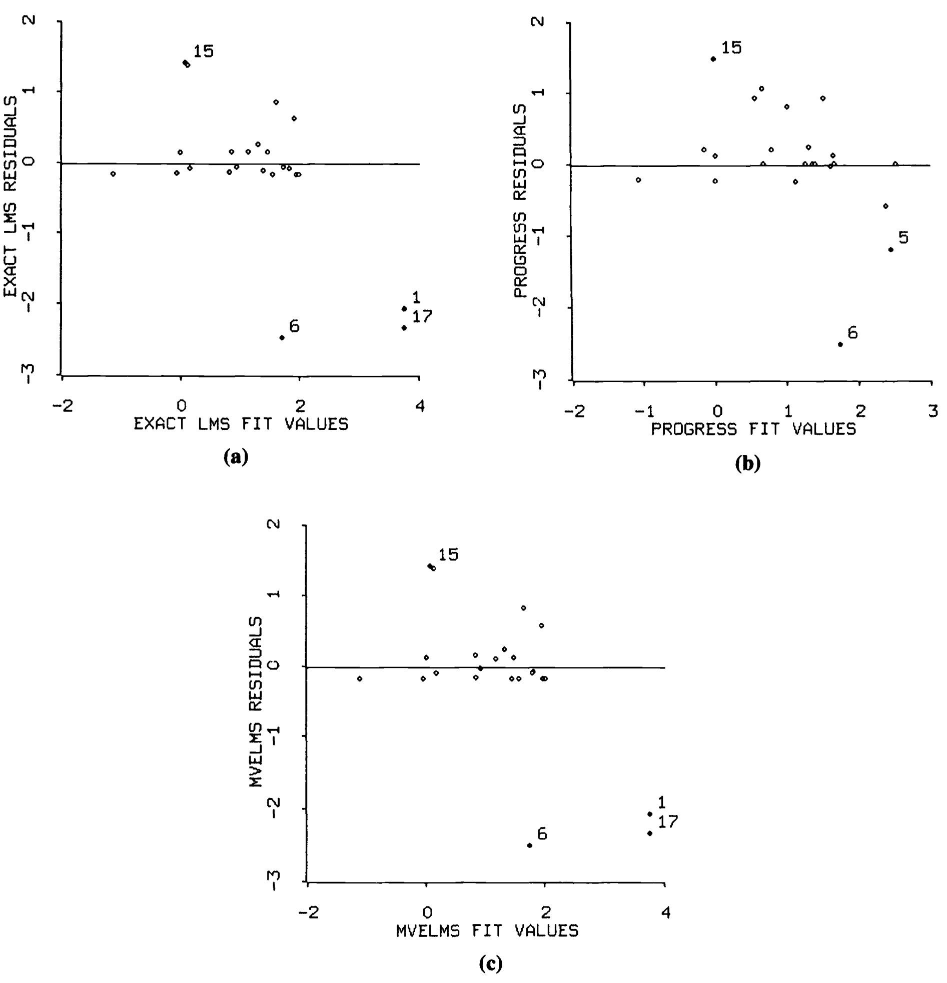
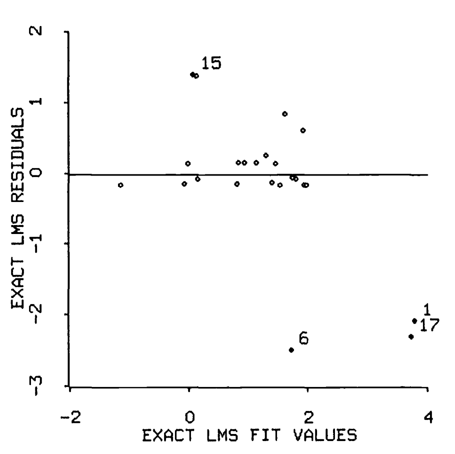


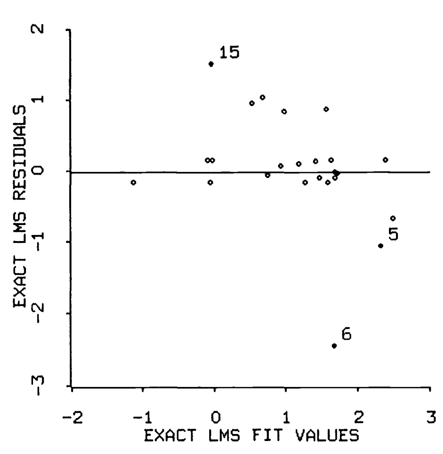
FIG. 3.2. LMS *residuals versus fit values for the cloud seeding data using*: (a) *Exact* LMS, (b) PROGRESS, *and* (c) MVELMS.

3.3. Engine knock data. This data set [10] has been discussed in several recent papers on high breakdown estimators. Hettmansperger and Sheather [9] and Ruppert [14] point out that a small recording error in the second point ("air" = 15.1 instead of 14.1) resulted in a vastly different LMS residual analysis than the correct data. Because they use the PROGRESS algorithm, it is possible that this instability was due to the use of an approximate algorithm. As discussed in Ruppert [14], the exact LMS algorithm is not sensitive to the recording error. To continue the discussion of the data set, we can compute the stability diagnostics suggested in this paper along with the exact LMS fit for the data with the recording error. Table 3.2 contains the residuals from the estimates that minimize the ninth through twelfth smallest residuals. Note that all the estimators yield similar residuals and thus similar analyses; therefore we can conclude that the LMS estimate is stable over the range of q's considered.

To further consider the stability of the LMS fit, we can consider the residuals from the exact LMS fit to 15 point subsets of the data. Table 3.3 presents residuals from these fits with various points deleted. The instability in these residuals can be considered a



(a)



(b)

FIG. 3.3. *Exact* LMS *residuals versus fit values for the cloud seeding data with point* 4 *at* (a) 1.1061 and (b) 1.1161.

TABLE 3.2
Engine knock residuals at LMS *estimate for range of values of q.*

Point\q	9	10	11	12
1	0.09	0.19	0.15	0.55
2	−3.13	−3.43	−4.02	−2.22
3	−0.09	−0.19	−0.34	0.82
4	−0.03	−0.07	−0.45	−0.82
5	5.91	6.05	6.66	6.21
6	0.09	0.10	0.45	0.69
7	−1.46	−1.25	−0.45	−0.82
8	−0.09	−0.01	0.45	−0.02
9	1.31	1.59	1.72	0.82
10	0.08	0.19	−0.02	0.14
11	−0.09	0.16	0.26	−0.59
12	−0.50	−0.19	−0.07	−0.82
13	2.90	3.14	3.48	3.36
14	0.09	−0.19	−0.45	−0.82
15	3.24	3.55	3.98	3.57
16	0.09	0.19	0.45	−0.16

lack of stability in the LMS fit for the entire data set; thus conclusions based on the LMS fit are questionable at best.

The same diagnostics were computed for the data without the recording error. Interestingly, no instability was detected. The LMS residuals for 15 point subsets of the data and the LQS estimate residuals were fairly stable.

3.4. Salinity data. This data was analyzed by Ruppert and Carroll [15]. It has $n = 28$ and $p = 4$, and computation time for the exact LMS fit is about 12 minutes. One of their conclusions was that the third and sixteenth points masked the influence of point 5, i.e., after those points were deleted the influence of point 5 was evident. Rousseeuw and Leroy [12] show that the PROGRESS approximation to the LMS estimate flags points 5, 8, 16, 23, and 24 as outliers and thus detects the influence of point 5. If the LMS estimate is unstable for this data set, then the detection of point 5 by Rousseeuw and Leroy was just chance. On the other hand, if the LMS estimate is fairly stable, their claim that the LMS estimate detected masking is reasonable in this case. Table 3.4 presents the

TABLE 3.3
Engine knock residuals at LMS *estimate with various points deleted.*

Point\ − i	1	4	7	11
1	0.15	−0.22	0.19	−0.22
2	−4.02	−0.22	−3.43	−0.22
3	−0.34	0.22	−0.19	0.22
4	−0.45	4.18	−0.07	4.18
5	6.66	5.02	6.05	5.02
6	0.45	0.10	0.10	0.10
7	−0.45	−4.19	−1.25	−4.19
8	0.45	0.22	−0.01	0.22
9	1.72	0.22	1.59	0.22
10	−0.02	−0.14	0.19	−0.14
11	0.26	−0.85	0.16	−0.85
12	−0.07	−2.42	−0.19	−2.42
13	3.48	−0.12	3.14	−0.12
14	−0.45	5.69	−0.19	5.69
15	3.98	−0.17	3.55	−0.17
16	0.45	−0.22	0.19	−0.22

TABLE 3.4
Residuals at LQS *estimate for a range of values of q.*

Point\ q	14	15	16	17	18	19
1	−1.60	−1.75	−1.87	−1.57	−1.63	−1.47
2	0.00	−0.26	−0.31	−0.24	−0.51	−0.02
3	0.88	0.26	0.31	−0.40	−0.51	−0.20
4	0.64	0.17	0.05	0.31	0.51	0.56
5	5.80	4.84	5.33	2.52	1.31	2.80
6	0.21	−0.16	−0.31	0.21	0.33	0.52
7	0.13	−0.09	−0.28	0.40	0.49	0.63
8	−2.48	−2.50	−2.74	−1.80	−1.69	−1.67
9	1.87	1.84	1.73	1.63	1.83	1.33
10	2.06	1.95	2.16	0.40	−0.19	−0.05
11	1.95	1.68	2.00	−0.22	−1.12	−0.54
12	−0.21	−0.24	−0.31	−0.40	−0.51	−0.54
13	1.14	1.19	1.07	1.27	1.19	1.15
14	0.21	0.26	0.31	−0.40	−0.97	−0.63
15	−0.14	−0.26	−0.04	−1.93	−2.48	−2.45
16	13.96	12.74	13.82	7.77	5.32	7.71
17	−0.21	−0.50	−0.31	−1.78	−2.37	−1.94
18	0.01	−0.12	−0.30	0.26	0.40	0.33
19	0.07	0.09	−0.08	0.11	0.43	−0.19
20	0.21	0.26	0.22	−0.24	−0.29	−0.63
21	0.07	0.24	0.10	0.33	0.26	0.09
22	−0.04	0.17	0.11	−0.02	−0.33	−0.32
23	4.79	4.51	5.02	1.61	0.42	0.99
24	4.10	3.73	4.21	1.06	−0.12	0.63
25	1.09	1.04	1.07	0.34	0.07	0.06
26	−0.19	−0.06	−0.18	−0.13	−0.17	−0.42
27	−0.17	0.05	−0.07	0.03	−0.07	−0.28
28	1.26	1.44	1.44	0.96	0.51	0.63

residuals from a range of LQS estimates. Although the $q = 18$ residuals are somewhat di erent from the others, the residuals are fairly stable over the range of q's considered. Note that, although reasonably stable, the residuals for points 8, 23, and 24 in Table 3.3 are not large for most of the q's; thus they should probably not be considered outliers. Points 5 and 16 appear to be stable LMS outliers. Consideration of the LMS fits to 27 point subsets of the data yields similar conclusions and detects no additional instability; thus this data set appears to be one in which the LMS estimate truly detects masking.

Remark. The first three examples were chosen to show that the suggested diagnostics can detect instability in the LMS estimate when it is presented. The last example is one in which the LMS estimate is stable and thus provides useful information. The exact LMS fit and the proposed diagnostics have been computed for many of the data sets found in [12]. In most cases, the LMS estimate was stable; thus it appears that the argument by Rousseeuw and Leroy that LMS is a useful statistical procedure is valid. It is the investigator's responsibility to check the stability of the LMS estimate for any given data set before making inferences based on the LMS fit.

Acknowledgments. The author thanks Douglas Hawkins, Jeff Simonoff, Simon Sheather, and David Ruppert for many helpful comments.

REFERENCES

[1] A. C. Atkinson, *Masking unmasked*, Biometrika, 73 (1986), pp. 533–542.

[2] E. W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

[3] R. D. Cook and D. M. Hawkins, *Comment on unmasking multivariate outliers and leverage points*, J. Amer. Statist. Assoc., 85 (1990), pp. 640–644.

[4] R. D. Cook and S. Weisberg, *Residuals and Influence in Regression*, Chapman-Hall, London, 1982.

[5] D. L. Donoho and P. J. Huber, *The notion of breakdown point*, in A Festschrift for Erich L. Lehmann, P. J. Bickel, K. A. Doksum, and J. L. Hodges, Jr., eds., Wadsworth, Belmont, CA, 1982, pp. 157–184.

[6] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1982.

[7] D. M. Hawkins, *A fast exact probabalistic algorithm for least median of squares regression*, Comput. Statist. Data Anal., 1993, to appear.

[8] D. M. Hawkins and J. S. Simonoff, *High breakdown and regression and multivariate estimation*, Appl. Statist., 42 (1993), pp. 423–432.

[9] T. P. Hettmansperger and S. J. Sheather, *A cautionary note on the method of least median squares*, Amer. Statist., 46 (1992), pp. 79–83.

[10] R. L. Mason, R. F. Gunst, and J. L. Hess, *Statistical Design and Analysis of Experiments*, John Wiley & Sons, New York, 1989.

[11] P. J. Rousseeuw, *Least Median of Squares Regression*, J. Amer. Statist. Assoc., 79 (1984), pp. 871–880.

[12] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, John Wiley and Sons, New York, 1987.

[13] P. J. Rousseeuw and B. C. van Zomeren, *Unmasking multivariate outliers and leverage points*, J. Amer. Statist. Assoc., 85 (1990), pp. 633–639.

[14] D. Ruppert, *Computing S-estimators for regression and multivariate location/dispersion*, J. Comput. and Graphical Statist., 1 (1992), pp. 253–270.

[15] D. Ruppert and R. J. Carroll, *Trimmed least squares estimation in the linear model*, J. Amer. Statist. Assoc., 75 (1980), pp. 828–838.

[16] D. G. Simpson, D. Ruppert, and R. Carroll, *One-step* GM-*estimates for regression with bounded influence and high breakdown point*, J. Amer. Statist. Assoc., 87 (1992), pp. 439–450.

[17] J. M. Steele and W. L. Steiger, *Algorithms and complexity for least median of squares regression*, Discrete Appl. Math., 14 (1986), pp. 93–100.

[18] P. Tichavsky, *Algorithms for and geometric characterizations of solutions in the* LMS *and* LTS *linear regression*, Comput. Statist. Quarterly, 2 (1991), pp. 139–151.

[19] C. F. J. Wu, *Jackknife, bootstrap and other resampling methods in regression analysis (with discussion)*, Ann. Statist., 14 (1986), pp. 1261–1350.

[20] V. J. Yohai, *High breakdown point and high efficiency robust estimates for regression*, Ann. Statist., 15 (1987), pp. 642–656.

[21] V. J. Yohai and R. Zamar, *High breakdown and point estimates of regression by means of the minimization of an efficient scale*, J. Amer. Statist. Assoc., 83 (1988), pp. 406–413.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1300–1319, November 1993

NUMERICAL AND ASYMPTOTIC SOLUTIONS FOR PERISTALTIC MOTION OF NONLINEAR VISCOUS FLOWS WITH ELASTIC FREE BOUNDARIES*

DALIN TANG† AND SAMUEL RANKIN†

Abstract. A mathematical model for peristaltic motion of nonlinear viscous flows with elastic free boundaries is introduced. An iterative numerical method is used to solve the free boundary problem. Long wave asymptotic expansion is developed and the zeroth order approximation is used as the numerical initial condition. The existence and uniqueness of the solution for the free boundary equation derived from the long wave expansion are proved. Computations were conducted to study the long wave approximation, the numerical solutions for the exact equations, and the influences of the parameters on the solutions.

Key words. peristaltic, Navier–Stokes, long wave, free boundary, elasticity

AMS subject classifications. 76, 39, 41, 34

1. Introduction. Peristaltic pumping, the physiological phenomenon of a circumferential progressive wave propagating along a flexible tube, plays an essential role in transporting fluid inside living organisms. Many modern mechanical devices have been designed on the principle of peristaltic pumping to transport fluids without internal moving parts, for example, the blood pump in the heart-lung machine and peristaltic transport of noxious fluid in the nuclear industry. Earlier mathematical work on the problem of peristaltic transport was based upon a viscous fluid model governed by the Navier–Stokes equations subject to a prescribed velocity on the boundary of the tube [11], [3]. A review of the research results can be found in the articles by Jaffrin and Shapiro [8] and Winet [21]. Numerical study of two-dimensional and axisymmetric peristaltic flows can be found in the articles by Takabatake, Ayukawa, and Mori [17], [18]. Recently, more refined models have been developed to deal with the peristaltic transport of a fluid-particle mixture or a heat-conducting fluid. The former was studied by Hung and Brown [7] and Kaimal [9], and the latter by Bestman [1] and Tang and Shen [19], [20].

In reality, the shape of the tube walls (e.g., blood vessels) is often unknown. They should be treated as free boundaries and solved as part of the solution. Experiments also suggest that the elastic properties of the tube walls should be taken into consideration [6], [10], [12]. In this paper, we introduce a three-dimensional (axisymmetric) model for viscous peristaltic motion with elastic free boundaries that combines three important factors: viscosity, elasticity, and free boundary. With the free boundary, this model should give a better representation of the actual physical situation than the fixed boundary models. Investigation of the free boundary model will provide useful information for designing equipment applying peristaltic motions and will lead to better understanding of some physiological processes involving peristalsis. However, the introduction of the free boundary makes this model difficult to solve. The fact that the domain is unknown makes it difficult to change the partial differential equation (PDE) system to a discretized difference system, which is the first step necessary to solve the PDE system using the finite difference method. To overcome this difficulty, we introduce a global iterative method for this model. The idea was originated from Fung [4]. To explain, we outline the method below.

Step 1. Obtain the long wave solution, which will be used as the numerical initial condition. The free boundary solution obtained from the long wave approximation will

*Received by the editors April 1, 1992; accepted for publication (in revised form) December 11, 1992. This research was supported in part by National Science Foundation grants DMS-9006043 and DMS-9209129.

†Mathematical Sciences Department, Worcester Polytechnic Institute, Worcester, Massachusetts 01609.

be used as the initial guess for the exact free boundary.

Step 2. With the boundary $\Gamma : r = H(x)$ obtained from Step 1 ($H(x)$ is the radius of the tube), use a local successive overrelaxation (SOR) method [20] to solve the system as a fixed boundary problem without the elasticity. The mapping

$$\xi = x, \tag{1.1}$$
$$\eta = r/H(x) \tag{1.2}$$

is used to map the (x, r) domain to a rectangular (ξ, η) domain. The number of iterations of the SOR method needed here should be determined by numerical experiment.

Step 3. Update the free boundary function $H(x)$ by using the elastic condition.

Step 4. With the newly updated $H(x)$, repeat Steps 2 and 3 until the desired accuracy is achieved. Adjust the number of local iterations if necessary to achieve the best convergence.

There are three key points in this method worth mentioning. (1) By using this procedure, the unknown domain becomes "known" at each global iteration and discretizing the PDE system becomes possible. (2) Using the long wave solution as the numerical initial condition is important to gain fast convergence. (3) The introduction of the mapping (1.1)–(1.2) makes the transformation of the (x, r) domain to a rectangular domain a fairly easy job.

In §2, we formulate the problem. The long wave asymptotic expansion is developed and solved in §3. The global iterative method is explained in §4. Results and discussions are given in §5.

2. Formulation.

We consider viscous flow in an elastic tube while the shape of the tube is to be determined. A tension function is prescribed on the boundary to reflect the elastic property of the tube wall. The tube and the motion are assumed to be axisymmetric and the wave traveling along the tube (x-direction) is periodic. By choosing a coordinate system moving with the wave, the boundary becomes stationary. The problem is formulated in Fig. 1.

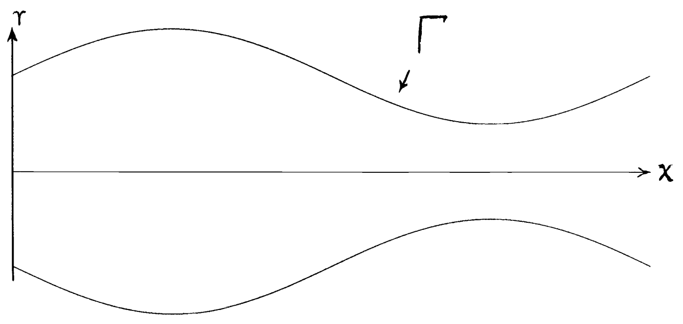


FIG. 1. *Peristaltic motion in an elastic tube.*

Equations of motion and continuity. Assuming the flow is Newtonian, viscous, and incompressible, we use the Navier–Stokes equations as the governing equations:

$$\rho \mathbf{u}_t + \rho(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mu \nabla^2 \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0, \quad t \geq 0, x \in \Omega,$$

where ρ is the density, $\mathbf{u}$ the velocity with respect to the moving coordinate system, p the pressure, μ the dynamic viscosity, and Ω is the domain consisting of one period of the tube. In terms of cylindrical coordinates, the nondimensionalized equations of motion and continuity with axisymmetry are

$$\frac{\partial v_x}{\partial t}+v_x\frac{\partial v_x}{\partial x}+v_r\frac{\partial v_x}{\partial r}=-\frac{\partial p}{\partial x}+\frac{1}{R}\left(\frac{\partial^2 v_x}{\partial x^2}+\frac{1}{r}\frac{\partial v_x}{\partial r}+\frac{\partial^2 v_x}{\partial r^2}\right), \tag{2.1}$$

$$\frac{\partial v_r}{\partial t}+v_x\frac{\partial v_r}{\partial x}+v_r\frac{\partial v_r}{\partial r}=-\frac{\partial p}{\partial r}+\frac{1}{R}\left(\frac{\partial^2 v_r}{\partial x^2}+\frac{\partial^2 v_r}{\partial r^2}+\frac{1}{r}\frac{\partial v_r}{\partial r}-\frac{v_r}{r^2}\right), \tag{2.2}$$

$$\frac{\partial v_x}{\partial x}+\frac{v_r}{r}+\frac{\partial v_r}{\partial r}=0, \tag{2.3}$$

where v_x and v_r are longitudinal and radial components of the velocity relative to the moving frame and R is the Reynolds number.

Free boundary. The free boundary $\Gamma : r = H(x)$ is to be determined as part of the solutions where $H(x)$ is the radius of the tube. We assume that $H(x)$ is periodic. Our result shows that for each prescribed initial opening

$$H(0) = H_0, \tag{2.4}$$

there is a solution to the free boundary equation obtained from the long wave approximation if certain conditions are met (see §3). This leads to the existence of the exact free boundary when the numerical method converges. However, the theoretical proof of the existence and uniqueness of the exact free boundary is a much harder problem and remains to be solved in the future.

Boundary conditions for the velocity and pressure. Considering boundary conditions, we assume no slipping between the fluid and wall, no penetration through the wall, and no horizontal motion of the wall. These lead to the following boundary conditions:

$$\mathbf{u}|_\Gamma = (v_x, v_r)|_\Gamma = (-C, f),$$

where Γ is the free boundary, C is the wave velocity, and f is determined by the radial motion of the free boundary. Recalling that the free boundary is $H = H(x) = H(x^* - Ct)$, where $H(x)$ is the radius of the free boundry and x^* is the old x-coordinate, we obtain

$$v_r|_\Gamma = \frac{dH}{dt} = \frac{\partial H}{\partial x^*}\frac{dx^*}{dt} + \frac{\partial H}{\partial t} = -CH'(x),$$

where the no-horizontal-motion condition $dx^*/dt = 0$ has been used. Now we have

$$\mathbf{u}|_\Gamma = (v_x, v_r)|_\Gamma = (-C, -CH'(x)). \tag{2.5}$$

It is easy to check that

$$\mathbf{u}\cdot\mathbf{n}|_\Gamma = 0,$$

i.e., the normal component of the velocity at the boundary is zero. Wave velocity C is prescribed with the tension function that is discussed below.

At $r = 0$, because of the symmetry, we assume

$$\frac{\partial v_x}{\partial r} = 0, \qquad v_r = 0.$$

At the two ends of the tube, we impose periodic conditions on the velocity and the pressure

$$v_x|_{x=0} = v_x|_{x=\ell'}, \tag{2.6}$$

$$v_r|_{x=0} = v_r|_{x=\ell'}, \tag{2.7}$$

$$p|_{x=0} = p|_{x=\ell'}, \tag{2.8}$$

where ℓ is the wave length. These periodic conditions are actually implied by the periodicity of the tension function and the Laplace law to be described below. Some numerical boundary conditions for the pressure at $r = 0$ and $\Gamma : r = H(x)$ will also be imposed. The details will be discussed later.

Additional boundary condition from elasticity—the Laplace law. Because the boundary is free, an additional boundary condition is needed to make the model complete. That condition comes from the consideration of the elastic property of the tube. Because of the complexity of structure of the tube walls in real application, there are many ways to introduce elasticity into a model. For simplicity, we will adopt the Laplace law to represent the elastic property of the wall [13]:

$$p|_\Gamma = \frac{T(x,r)}{r}, \tag{2.9}$$

where $T(x, r)$ is the prescribed tension function. Several cases are discussed in this paper. In practice, various functions can be introduced to find the best agreement with experimental data. When necessary, we may introduce more complicated elasticity laws involving stresses and strains of the walls, which would make the model more practical. The change of the elastic condition will affect only the part of updating the free boundary. Therefore, the numerical method will still be applicable with minor adjustments.

Remark. The prescribed tension is fundamental to the whole mechanism. We assume that it takes the form of a traveling wave

$$T = T(x^*, t, r) = T(x^* - Ct, r) = T(x, r),$$

with given wave speed C, period, and wave length. We are then looking for solutions in which the fluid velocity, pressure, and the free boundary also adopt the form of traveling waves with the same wave speed, period, and wave length as the imposed wave of elasticity.

From the Laplace law, it is clear that prescribing T is equivalent to prescribing the pressure at the free boundary.

Flux condition. Using $\mathbf{u} \cdot \mathbf{n}|_\Gamma = 0, \nabla \cdot \mathbf{u} = 0$, and the divergence theorem, we can prove [19]

$$\int_{A(x)} \mathbf{u} \cdot \mathbf{n} dA = \text{const} = Q,$$

where $A(x)$ is the cross section at x and Q is the flux. This is the so-called flux condition.

In terms of cylindrical coordinates, the flux condition can be written as

$$\int_{A(x)} \mathbf{u} \cdot \mathbf{n} dA = \int_0^{2\pi} \int_0^{H(x)} v_x \, r \, dr \, d\theta = Q$$

or

$$\int_0^{H(x)} v_x r dr = \frac{Q}{2\pi} = \text{const.} \tag{2.10}$$

For the fixed boundary model, it has been proved that for each prescribed flux, there exists a unique solution to the system [19]. A similar conclusion is also true for the fixed boundary model if we replace the flux condition by the pressure drop condition, i.e.,

$$p|_{x=\ell} - p|_{x=0} = \text{const} = P_d,$$

where P_d is the prescribed pressure drop. The relation between the flux and the pressure drop is almost linear for the fixed boundary model [20].

For the free boundary model, the situation is different. Due to the periodicity of the free boundary and the Laplace law just introduced, the pressure drop over one period of the tube must be zero. That means the pressure drop cannot be prescribed for the free boundary model. It is found in this paper that prescribing the flux is equivalent to prescribing the intial tube opening $H(0) = H_0$ for the free boundary model. For theoretical convenience, we choose to prescribe H_0.

Although we cannot prescribe the flux condition once H_0 is given, we still have that identity which will be used to derive the free boundary equation from the long wave approximation. The constant Q will be determined as part of the solution.

Remark. To prescribe pressure drop, tapering of the tube must be taken into consideration. This will be treated in a separate paper.

Remark. Recall that we are using a moving coordinate system. The laboratory longitudinal velocity u^* can be expresssed in terms of v_x by

$$u^* = v_x + C. \tag{2.11}$$

So the laboratory flux

$$Q^*(x^*) = \int_{A(x^*)} u^* dA = \int_{A(x^*)} v_x dA + \int_{A(x^*)} C dA = \text{const} + C \int_{A(x^*)} dA \tag{2.12}$$

will be a function of x^*, not a constant.

From the above, we have the mathematical model for the viscous flow with an elastic free boundary (in nondimensionalized version):

$$\begin{aligned}
&\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{R}\nabla^2 \mathbf{u}, && \text{(equation of motion)},\\
&\nabla \cdot \mathbf{u} = 0, && \text{(equation of continuity)},\\
&\mathbf{u}|_\Gamma = \left(-C, -C\frac{dH}{dx}\right), && \text{(boundary conditions for } \mathbf{u} \text{ at } \Gamma),\\
&\frac{\partial v_x}{\partial r}|_{r=0} = 0, \quad v_r|_{r=0} = 0, && \text{(boundary conditions for } \mathbf{u} \text{ at } r = 0),\\
&\mathbf{u}|_{x=0} = \mathbf{u}|_{x=\ell}, && \text{(periodic condition for } \mathbf{u}),\\
&p|_{x=0} = p|_{x=\ell}, && \text{(periodic condition for } p),
\end{aligned}$$

$$\Gamma : r = H(x), \qquad H(0) = H(\ell) = H_0, \text{ (conditions for the free boundary)},$$
$$p|\Gamma = \frac{T(x,r)}{r}, \qquad \text{(Laplace law)}.$$

Assuming axisymmetry, the stationary system can be expressed in terms of cylindrical coordinates as

$$v_x \frac{\partial v_x}{\partial x} + v_r \frac{\partial v_x}{\partial r} = -\frac{\partial p}{\partial x} + \frac{1}{R}\left(\frac{\partial^2 v_x}{\partial x^2} + \frac{1}{r}\frac{\partial v_x}{\partial r} + \frac{\partial^2 v_x}{\partial r^2}\right), \tag{2.13}$$

$$v_x \frac{\partial v_r}{\partial x} + v_r \frac{\partial v_r}{\partial r} = -\frac{\partial p}{\partial r} + \frac{1}{R}\left(\frac{\partial^2 v_r}{\partial x^2} + \frac{\partial^2 v_r}{\partial r^2} + \frac{1}{r}\frac{\partial v_r}{\partial r} - \frac{v_r}{r^2}\right), \tag{2.14}$$

$$\frac{\partial v_x}{\partial x} + \frac{v_r}{r} + \frac{\partial v_r}{\partial r} = 0, \tag{2.15}$$

$$(v_x, v_r)|_\Gamma = (-C, -CH'(x)), \tag{2.16}$$

$$\frac{\partial v_x}{\partial r}|_{r=0} = 0, \qquad v_r|_{r=0} = 0, \tag{2.17}$$

$$v_x|_{x=0} = v_x|_{x=\ell}, \qquad v_r|_{x=0} = v_r|_{x=\ell}, \tag{2.18}$$

$$p|_{x=0} = p|_{x=\ell}, \tag{2.19}$$

$$\Gamma : r = H(x), \quad H(0) = H(\ell) = H_0, \tag{2.20}$$

$$p|_\Gamma = \frac{T(x,r)}{r}, \qquad 0 \le x \le \ell, 0 < r \le H(x). \tag{2.21}$$

Compared with the fixed boundary model, this model treats the boundary as a free boundary. An additional boundary condition is introduced by using the Laplace law of elasticity. Instead of prescribing the flux condition, the initial opening of the tube $H_0 = H(0)$ is prescribed to ensure a unique solution of the system.

3. Long wave asymptotic approximation. Assume that $0 < d/\ell = \epsilon << 1$, where d is the average radius of the tube and ℓ is the wave length. Our previous experience indicates that $p = O(\epsilon^{-2})$. The Laplace law implies $T = O(p) = O(\epsilon^{-2})$. Then by assuming $R = O(\epsilon)$, the zeroth order long wave approximation will be essentially a one-dimensional linear system with free boundary; therefore, it is much easier to handle. Please note that although we made the assumption $R = O(\epsilon)$, it does not imply that our numerical method will be valid only for small Reynolds numbers since we are basically using this approximation as the numerical initial condition. The numerical method covered in §4 does apply to finite Reynolds number cases. For simplicity, when we simply replace, respectively, $x, \partial/\partial x, p, \ell, R, T$, by $x/\epsilon, \epsilon(\partial/\partial x), p\epsilon^{-2}, \ell/\epsilon, \epsilon R, T\epsilon^{-2}$ in (2.13)–(2.21), the system becomes

$$
\begin{aligned}
&\epsilon^2 v_x \frac{\partial v_x}{\partial x} + \epsilon v_r \frac{\partial v_r}{\partial r} = -\frac{\partial p}{\partial x} + \frac{1}{R}\left(\epsilon^2 \frac{\partial^2 v_x}{\partial x^2} + \frac{1}{r}\frac{\partial v_x}{\partial r} + \frac{\partial^2 v_x}{\partial r^2}\right),\\
&\epsilon^2 v_x \frac{\partial v_r}{\partial x} + \epsilon v_r \frac{\partial v_r}{\partial r} = -\epsilon^{-1}\frac{\partial p}{\partial r} + \frac{1}{R}\left(\epsilon^2 \frac{\partial^2 v_r}{\partial x^2} + \frac{\partial^2 v_r}{\partial z^2} + \frac{1}{r}\frac{\partial v_r}{\partial r} - \frac{v_r}{r^2}\right),\\
&\epsilon \frac{\partial v_x}{\partial x} + \frac{v_r}{r} + \frac{\partial v_r}{\partial r} = 0,\\
&(v_x, v_r)|_\Gamma = (-C, -C\epsilon H'(x)),\\
(3.1)\quad &\frac{\partial v_x}{\partial r}|_{r=0} = 0, \qquad v_r|_{r=0} = 0,\\
&v_x|_{x=0} = v_x|_{x=\ell}, \qquad v_r|_{x=0} = v_r|_{x=\ell},\\
&p|_{x=0} = p|_{x=\ell},\\
&\Gamma : r = H(x), \qquad H(0) = H(\ell) = H_0,\\
&p|_\Gamma = \frac{T(x,r)}{r}.
\end{aligned}
$$

Because of the asymptotic assumptions, the odd terms of the asymptotic expansions of $\mathbf{u}, p$, and H will turn out to be zero. Therefore, we assume

$$
\begin{aligned}
\mathbf{u} &= \mathbf{u}_0 + \epsilon^2 \mathbf{u}_2 + \epsilon^4 \mathbf{u}_4 + \cdots,\\
p &= p_0 + \epsilon^2 p_2 + \epsilon^4 p_4 + \cdots,\\
H &= H_0 + \epsilon^2 H_2 + \epsilon^4 H_4 + \cdots,
\end{aligned}
$$

where $\mathbf{u}_i = (v_x^i, v_r^i)$. Substituting these into (3.1) for the zeroth order approximation, we obtain (in the following, we use H for $H_0(x)$ and H_0 for the initial opening),

$$\frac{1}{R}\left(\frac{1}{r}\frac{\partial v_x^0}{\partial r} + \frac{\partial^2 \partial v_x^0}{\partial r^2}\right) = \frac{\partial p_0}{\partial x}, \tag{3.2}$$

$$\frac{\partial p_0}{\partial r} = 0, \tag{3.3}$$

$$\frac{\partial v_r^0}{r} + \frac{\partial v_r^0}{\partial r} = 0, \tag{3.4}$$

$$(v_x^0, v_r^0)|_\Gamma = (-C, 0), \tag{3.5}$$

$$\frac{\partial v_x^0}{\partial r}|_{r=0} = 0, \qquad v_r^0|_{r=0} = 0, \tag{3.6}$$

$$v_x^0|_{x=0} = v_x^0|_{x=\ell}, \qquad v_r^0|_{x=0} = v_r^0|_{x=\ell}, \tag{3.7}$$

$$p_0|_{x=0} = p_0|_{x=\ell}, \tag{3.8}$$

$$\Gamma : r = H(x), \quad H(0) = H(\ell) = H_0, \tag{3.9}$$

$$p_0|_\Gamma = \frac{T(x,r)}{r}, \tag{3.10}$$

while

$$\int_0^H v_x^0 r dr = \frac{Q}{2\pi} \tag{3.11}$$

is an identity we will need in deriving the free boundary equation.

From (3.3), we see that $p_0 = p_0(x)$. Using (3.2) and the corresponding boundary conditions, we obtained

$$v_x^0 = \frac{1}{4}R(r^2 - H^2)\frac{\partial p_0}{\partial x} - C. \tag{3.12}$$

Plug into (3.11) and integrate

$$-\frac{R}{8}\frac{\partial p_0}{\partial x}H^4 - CH^2 = \frac{Q}{\pi}. \tag{3.13}$$

From (3.10), p_0 can be expressed in terms of $H(x)$ as

$$p_0 = \frac{T(x, H(x))}{H(x)}.$$

Thus (3.13) contains one unknown function $H(x)$ only. If $H(x)$ can be solved from (3.13) and (3.9), then p_0 and v_x^0 are also determined. It is easy to see from (3.4)–(3.6) that

$$v_r^0 = 0.$$

We consider the three cases below that are chosen because they are the three easiest simplifications of the general function $T(x, r)$. Comparison between the numerical and experimental results must be done to see how realistic these conditions are.

Case 1. $T(x, r) = T(r)$. From (3.10), $p_0 = (T(H))/(H) = p_0(H)$. Using this information and also letting $\ell = 1$ for simplicity, (3.13) and (3.9) become

$$-\frac{R}{8}\frac{\partial p_0}{\partial H}\frac{dH}{dx}H^4 - CH^2 = \frac{Q}{\pi}, \qquad H(0) = H(1) = H_0. \tag{3.14}$$

Equation (3.14) has a constant solution

$$H = \left(-\frac{Q}{(\pi C)}\right)^{1/2}. \tag{3.15}$$

It follows from here that

$$p_0 = \text{const}, \tag{3.16}$$

$$v_x^0 = -C, \tag{3.17}$$

$$v_r^0 = 0. \tag{3.18}$$

We also solved (3.14) numerically and (3.15) was the only solution we found. It turns out that (3.15)–(3.18) is also the exact solution to (2.13)–(2.21). Therefore, we suspect that the constant solution is the only solution to the system. However, we cannot yet provide a theoretical proof.

Case 2. $T(x,r) = T_0 b(x)$, where $b(x)$ is continuously differentiable and periodic with period $1, 0 < r_1 \leq b(x) \leq r_2 < \infty$. Now we have

$$p_0 = \frac{T_0 b(x)}{H}, \qquad p_{0x} = \frac{T_0 b'(x) H - T_0 b(x) H'}{H^2}.$$

The free boundary equation is

$$(3.19) \quad H'(x) = \frac{b'}{b} H + \frac{8}{RT_0}\left(\frac{C}{b(x)} + \frac{Q}{\pi b(x) H^2(x)}\right), \qquad H(0) = H(1) = H_0.$$

For (3.19), we have the following theorem.

THEOREM. *Let $b(x) \in C^1[0,1]$ be periodic with period 1, $b(0) = b(1) = 1$, $0 < b_1 \leq b(x) \leq b_2 < \infty$, $C \cdot Q \leq 0$, $\epsilon = (8/RT_0)$. Then for each $H_0 > 0$, there is $\epsilon_0 > 0$, such that for $\epsilon < \epsilon_0$, there exists a Q such that the free boundry equation* (3.19) *has a unique solution.*

Proof. We need the following lemma.

LEMMA ([2, Thm. 15.1, p. 148]). *Let $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ be Banach spaces, and let $\mathbf{U} \subset \mathbf{X}$, and $\mathbf{V} \subset \mathbf{Y}$ be neighborhoods of x_0 and y_0, respectively. Let $\mathcal{F}(x,y) : \mathbf{U} \times \mathbf{V} \to \mathbf{Z}$ be continuous and continuously differentiable with respect to y. Suppose also that $\mathcal{F}(x_0, y_0) = 0$ and $\mathcal{F}_y^{-1}(x_0, y_0) \in \mathbf{L(Z,Y)}$. Then there exist balls $\overline{\mathbf{B}}_{\delta_x}(x_0) \subset \mathbf{U}$, $\overline{\mathbf{B}}_{\delta_y}(y_0) \subset \mathbf{V}$ and exactly one map $\mathcal{T} : \mathbf{B}_{\delta_x}(x_0) \to \bar{B}_{\delta_y}(y_0)$ such that $\mathcal{T} x_0 = y_0$ and $\mathcal{F}(x, \mathcal{T} x) = 0$ on $\mathbf{B}_{\delta_x}(x_0)$. This map $\mathcal{T}$ is continuous.* □

Proof of the theorem. Let $f(x) = 1/H(x)$, (3.16) is changed to

$$(bf)' = -\epsilon\left(Cf^2 + \frac{Q}{\pi} f^4\right).$$

Integrate from 0 to x,

$$b(x) f(x) = f(0) - \epsilon \int_0^x \left(Cf^2 + \frac{Q}{\pi} f^4\right) dx,$$

where $f(0) = 1/H(0)$. $f(0) = f(1)$ implies

$$(3.20) \qquad Q = -\pi C \frac{\int_0^1 f^2 dx}{\int_0^1 f^4 dx}.$$

The equation now becomes a nonlinear integral equation

$$(3.21) \qquad b(x) f(x) - f(0) + \epsilon C \int_0^x \left(f^2 - \frac{\int_0^1 f^2 dx}{\int_0^1 f^4 dx} f^4\right) dx = 0.$$

Introducing

$$(3.22) \qquad g(x) = f(x) - \frac{f(0)}{b(x)},$$

$$(3.23) \qquad \mathcal{L} g = b(x) f(x) - f(0) = b(x) g(x),$$

$$\begin{aligned}\mathcal{N}g &= C\int_0^x \left(f^2 - \frac{\int_0^1 f^2dx}{\int_0^1 f^4dx} f^4\right) dx \\ &= C\int_0^x \left(\left(g(x)+\frac{1}{bH_0}\right)^2 - \frac{\int_0^1 \left(g+\frac{1}{bH_0}\right)^2 dx}{\int_0^1 \left(g+\frac{1}{bH_0}\right)^4 dx}\left(g+\frac{1}{bH_0}\right)^4\right) dx, \qquad (3.24)\\ g\in \mathbf{Y} &= \{g|g(x)\in C[0,1], g(0)=g(1)=0\},\end{aligned}$$

then $\mathcal{L}:\mathbf{Y}\to\mathbf{Y}$ is one-to-one and onto, $\mathcal{L}^{-1}$ exists. Equation (3.20) becomes

$$\mathcal{L}g + \epsilon\mathcal{N}g = 0. \qquad (3.25)$$

Define

$$\mathcal{F}(\epsilon, g) = \mathcal{L}g + \epsilon\mathcal{N}g.$$

Then $\mathcal{F}$ is an operator from $\mathbf{R}\times\mathbf{Y}$ to $\mathbf{Y}$. It is easy to check that

$$\mathcal{F}(0,0)=0, \quad \mathcal{F}_g(0,0)=\mathcal{L}, \quad \mathcal{F}_g^{-1}(0,0)=\mathcal{L}^{-1}.$$

Then, by the lemma, there exist balls $\mathbf{B}_{\epsilon_0}(0)\subset R$, $\mathbf{B}_{\delta_y}(0)\subset\mathbf{Y}$, and a unique mapping $\mathcal{T}:\mathbf{B}_{\epsilon_0}(0)\to\mathbf{B}_{\delta_y}(0)$ such that

$$\mathcal{F}(\epsilon,\mathcal{T}\epsilon)=0 \quad \text{for } \epsilon<\epsilon_0.$$

Furthermore, the mapping $\mathcal{T}$ is continuous, i.e., for δ small, we can choose ϵ such that

$$|\mathcal{T}_\epsilon| = |g(x)| \le \delta.$$

We choose δ small so that

$$f(x) = g(x) + \frac{1}{b(x)H_0} \ge d_1 > 0.$$

Then

$$H(x) = \frac{1}{g(x)+\frac{1}{b(x)H_0}} = \frac{H_0 b(x)}{H_0 b(x) g(x)+1} \qquad (3.26)$$

is the solution to the free boundary equation (3.19) and Q is given by (3.20). The proof is complete.

Case 3. $T(x,r)=T(r)b(x)$, where $b(x)$ is the same as in Case 2. For simplicity, let $T(r)=T_0+T_1r$. Similar calculation leads to

$$\frac{dH}{dx} = \frac{b'}{b}H + \frac{1}{T_0}\left(\frac{b'}{b}T_1H^2 + \frac{8C}{Rb} + \frac{8Q}{\pi RbH^2}\right). \qquad (3.27)$$

Equation (3.27) is similar to (3.19). By using the same procedure, similar results can be proved. We omitted the details here.

Remark. Although we have the existence and uniqueness of the free boundary for a long wave free boundary such as (3.19), we do not yet have the corresponding result for the exact system.

4. Numerical method for the exact system. The method is outlined in §2. As a numerical example, let $T = T_0(1 + a\sin 2\pi\alpha x)$, $0 \le x \le 1/\alpha$, where $\alpha = 1/\ell$ is the long wave parameter and ℓ is the wave length. Here we have assumed that the average radius of the tube $d = O(1)$ (Note: d is unknown.). The system to be solved is

$$(4.1)\quad \begin{aligned}
&v_x\frac{\partial v_x}{\partial x} + v_r\frac{\partial v_x}{\partial r} = -\frac{\partial p}{\partial x} + \frac{1}{R}\left(\frac{\partial^2 v_x}{\partial x^2} + \frac{1}{r} - \frac{\partial v_x}{\partial r} + \frac{\partial^2 v_x}{\partial r^2}\right),\\
&v_x\frac{\partial v_r}{\partial x} + v_r\frac{\partial v_r}{\partial r} = -\frac{\partial p}{\partial r} + \frac{1}{R}\left(\frac{\partial^2 v_r}{\partial x^2} + \frac{\partial^2 v_r}{\partial r^2} + \frac{1}{r}\frac{\partial v_r}{\partial r} - \frac{v_r}{r^2}\right),\\
&\frac{\partial v_x}{\partial x} + \frac{v_r}{r} + \frac{\partial v_r}{\partial r} = 0.\\
&(v_x, v_r)|_\Gamma = (-C, -CH'(x)),\\
&\frac{\partial v_x}{\partial r}|_{r=0} = 0, \qquad v_r|_{r=0} = 0,\\
&\mathbf{u}|_{x=0} = \mathbf{u}|_{x=\ell},\\
&p|_{x=0} = p|_{x=\ell},\\
&\Gamma : r = H(x), \qquad H(0) = H(\ell) = H_0,\\
&p|_\Gamma = \frac{T_0(1 + a\sin 2\pi\alpha x)}{H}, \quad 0 < x < \ell, \quad 0 < r < H(x).
\end{aligned}$$

Step 1. Obtain the long wave approximation. In terms of the new parameters introduced during the long wave equation derivation, the free boundary equation is

$$(4.2)\qquad \frac{dH}{dx} = \left[(2\pi a\cos 2k\pi x)H + \frac{8Q}{\pi RT_0H^2} + \frac{8C}{RT_0}\right]/[1 + a\sin 2\pi x],$$

$$(4.3)\qquad H(0) = H(1) = H_0,$$

where H_0 is the prescribed initial radius, C the wave velocity, and Q is to be determined with the solution. Equations (4.2)–(4.3) are solved numerically to get H_x. Back to the original parameters and variables (indicated by *), the pressure and velocity obtained from the long wave approximation are given by

$$(4.4)\qquad p_0^* = \frac{T_0^*(1 + a\sin 2\pi\alpha x^*)}{H},$$

$$(4.5)\quad v_x^{0^*} = \frac{\alpha R^*}{4}(r^2 - H^2)\frac{T_0^* 2\pi a\cos 2\pi\alpha x^* H - T_0^*(1 + a\sin 2\pi\alpha x^*)dH/dx}{H^2} - C,$$

$$(4.6)\qquad v_r^{0^*} = 0,$$

where dH/dX is given by

$$(4.7)\quad \frac{dH}{dx} = \left[(2\pi a\cos 2\pi\alpha x^*)H + \frac{8\alpha Q}{\pi R^* T_0^* H^2} + \frac{8C\alpha}{R^* T_0^*}\right]/[1 + a\sin 2\pi\alpha x^*].$$

Step 2. With the $H(x)$ obtained from Step 1, solve the fixed boundary problem on the domain $0 \le x \le 1/\alpha, 0 \le r \le H(x)$. Using (1.1)–(1.2), $\xi = x, \eta = r/H(x)$, we can transform the domain $0 \le x \le 1/\alpha$, $0 \le r \le H(x)$ to $0 \le \xi \le 1/\alpha$, $0 \le \eta \le 1$. In computing derivatives, the following formulas are useful:

$$\begin{aligned} f_x &= f_\xi + f_\eta \eta_x, \\ f_r &= f_\eta \eta_r, \\ f_{xx} &= f_{\xi\xi} + 2f_{\xi\eta}\eta_x + f_{\eta\eta}\eta_x^2 + f_\eta \eta_{xx}, \\ f_{rr} &= f_{\eta\eta}\eta_r^2, \\ f_{xx} + f_{rr} &= f_{\xi\xi} + 2f_{\xi\eta}\eta_x + f_{\eta\eta}(\eta_x^2 + \eta_r^2) + f_\eta \eta_{xx}, \end{aligned} \tag{4.8}$$

where

$$\eta_x = -\frac{rH'(x)}{H^2}, \quad \eta_r = \frac{1}{H(x)}, \quad \eta_{xx} = -\frac{rHH'' - 2rH'^2}{H^3}.$$

Using the notation (u, v) for (v_x, v_r), the system in terms of (ξ, η) assumes the form

(4.9)

$$\begin{aligned} u(u_\xi + u_\eta \eta_x) + v u_\eta \eta_r &= -(p_\xi + p_\eta \eta_x) \\ &\quad + \frac{1}{R}\left(u_{\xi\xi} + 2u_{\xi\eta}\eta_x + u_{\eta\eta}(\eta_x^2 + \eta_r^2) + u_\eta \eta_{xx} + \frac{1}{r}u_\eta \eta_r\right), \end{aligned}$$

(4.10)

$$\begin{aligned} u(v_\xi + v_\eta \eta_x) + v v_\eta \eta_r &= -p_\eta \eta_r \\ &\quad + \frac{1}{R}\left(v_{\xi\xi} + 2v_{\xi\eta}\eta_x + v_{\eta\eta}(\eta_x^2 + \eta_r^2) + v_\eta \eta_{xx} + \frac{1}{r}v_\eta \eta_r - \frac{v}{r^2}\right), \end{aligned}$$

$$u_\xi + u_\eta \eta_x + \frac{v}{r} + v_\eta \eta_r = 0, \tag{4.11}$$

$$(u, v)|_\Gamma = \left(-C, -C\frac{dH}{d\xi}\right), \tag{4.12}$$

$$u_\eta|_{r=0} = 0, \qquad v|_{r=0} = 0, \tag{4.13}$$

$$u|_{x=0} = u|_{x=\ell}, \qquad v|_{x=0} = v|_{x=\ell}, \tag{4.14}$$

$$p|_{x=0} = p|_{x=\ell}, \tag{4.15}$$

$$\Gamma : \eta = 1, \qquad 0 \le \xi \le \frac{1}{\alpha}, \tag{4.16}$$

$$p|_\Gamma = \frac{T_0(1 + a \sin 2\pi\alpha\xi)}{H}, \tag{4.17}$$

where u, v, p, and the free boundary $H(x)$ are all periodic in ξ with period $1/\alpha$.

We use the regularized central difference scheme and the extended successive over-relaxation (ESOR) iterative method suggested by Strikwerda [14], [15] to solve this fixed boundary problem. The method has been used by the author successfully in [20] as it is relatively easy to program, is of second-order accuracy, and provides good convergence. The finite difference scheme used here is briefly explained below. Let d_1 and d_2 be the spans of finite differences for ξ, η, respectively. We use the following formulas for the derivatives to convert the differential equations into finite difference equations.

$$\begin{aligned}
f(i,j) &= f(\xi_i,\eta_j) = f(i\cdot d_1, j\cdot d_2).\\
f_{\xi\xi}(i,j) &= [f(i+1,j)+f(i-1,j)-2f(i,j)]/d_1^2,\\
f_{\eta\eta}(i,j) &= [f(i,j+1)+f(i,j-1)-2f(i,j)]/d_2^2,\\
f_{\xi\eta}(i,j) &= [f(i+1,j+1)-f(i-1,j+1)-f(i+1,j-1)+f(i-1,j-1)]/(4d_1d_2),\\
f_\xi(i,j) &= \delta_{\xi o}f-(1/6)d_1^2\delta_{\xi-}\delta_{\xi+}^2 f,\\
f_\eta(i,j) &= \delta_{\eta o}f-(1/6)d_2^2\delta_{\eta-}\delta_{\eta+}^2 f,
\end{aligned}$$

where

$$\begin{aligned}
(\delta_{\xi o}f)(i,j) &= [f(i+1,j)-f(i-1,j)]/(2d_1),\\
(\delta_{\xi+}f)(i,j) &= [f(i+1,j)-f(i,j)]/d_1,\\
(\delta_{\xi-}f)(i,j) &= [f(i,j)-f(i-1,j)]/d_1,
\end{aligned}$$

and the corresponding differences for η can be defined similarly. The third-order terms in the first difference formulas are necessary to have a regular scheme. The iterative scheme is given below:

(4.18)

$$\begin{aligned}
u^*(i,j) &= u(i,j)\\
&-\omega\left\{u(i,j) - \left[\frac{u(i+1,j)+u(i-1,j)}{d_1^2} + \frac{u(i,j+1)+u(i,j-1)}{d_2^2}(\eta^2+\eta_r^2)\right.\right.\\
&\qquad + 2u_{\xi\eta}\eta_x + u_\eta\eta_{xx} - R(u(u_\xi+u_\eta\eta_x)+vu_\eta\eta_r)\\
&\qquad \left.\left. - R(p_\xi+p_\eta\eta_x)+u_\eta\eta_r/r\right] \Big/ \left[\frac{2}{d_1^2}+\frac{2}{d_2^2}(\eta_x^2+\eta_r^2)\right]\right\},
\end{aligned}$$

(4.19)

$$\begin{aligned}
v^*(i,j) &= v(i,j)\\
&-\omega\left\{v(i,j) - \left[r^2\left(\frac{v(i+1,j)+v(i-1,j)}{d_1^2} + \frac{v(i,j+1)+v(i,j-1)}{d_2^2}(\eta_x^2+\eta_r^2)\right)\right.\right.\\
&\qquad \left. + r^2(2v_{\xi\eta}\eta_x+v_\eta\eta_{xx}) - Rr^2(u(v_\xi+v_\eta\eta_x)+vv_\eta\eta_r+p_\eta\eta_r)+rv_\eta\eta_r\right]\\
&\qquad \left. \Big/ \left[r^2\left(\frac{2}{d_1^2}+\frac{2}{d_2^2}(\eta_x^2+\eta_r^2)\right)+1\right]\right\},
\end{aligned}$$

$$p^*(i,j) = p(i,j) - \gamma\{r(u_\xi + u_\eta \eta_x) + v + rv_\eta \eta_r\}, \tag{4.20}$$

where u^*, v^*, and p^* are the updated values of the corresponding functions, and ω and γ are iteration constants. We used both finite differences and derivatives in (4.18)–(4.20) to shorten the formulas. When computing, those derivatives were calculated first and then (4.18)–(4.20) were performed.

We impose periodic boundary conditions on u, v, and p in ξ-direction. At $\eta = 0$, cubic interpolation was used for pressure, e.g.,

$$p(i,0) = 3(p(i,1) - p(i,2)) + p(i,3). \tag{4.21}$$

Cubic interpolation was also used for pressure at $\eta = 1$. Boundary condition (4.12) was used for (u, v) at $\eta = 1$. At $\eta = 0$, using second-order difference, (4.13) implies

$$u(i,0) = (4u(i,1) - u(i,2))/3, \tag{4.22}$$

$$v(i,0) = 0. \tag{4.23}$$

For computation, our experience indicates that $\omega = 0.001$—1.5 and $\gamma = 0.1\omega$ give good convergence. When the Reynolds number R is small, we chose $\omega = 1.5$, $\gamma = 0.1$. For larger $R (100 \leq R \leq 2000)$, we chose smaller ω and γ to make the algorithm converge.

Remark. The PDE needs two boundary conditions at $\eta = 0, 1$, and those conditions are given by (4.12)–(4.14). The cubic interpolations for the pressure at $\eta = 0, 1$ are numerical boundary conditions and do not make the system overdetermined. For reference on this regard, see [16, p. 298].

Step 3. We use the long wave approximation as the first guess. After a few local iterations for the fixed boundary problem, the boundary is updated according to (4.17):

$$H^*(\xi) = \frac{T_0(1 + a \sin 2\pi\alpha\xi)}{p(\xi, \eta)}\Big|_{\eta=1}.$$

Then the transformation (1.1)–(1.2) is modified using the new H^* and $\eta_x, \eta_r, \eta_{xx}$ are updated. This is one global iteration.

Step 4. Repeat Steps 2 and 3 as many times as needed. Our experience indicates that for most cases the number of local iterations is around 20. For some cases, we can achieve convergence by adjusting the number several times between 10 and 100 at the beginning stage of the computation.

Remark. "Convergence" is used here in the sense that the corrections to the numerical solutions made at each iteration, or, equivalently, the imbalances of the equations when the numerical solutions are plugged in will become and remain small after some iterations. (The imbalances are the L_2 norms of the parts inside $\{\cdots\}$ in (4.18)–(4.20). The relative imbalances are the above imbalances divided by the L_2 norms of u, v, and p, respectively.) The theoretical justification of the numerical method will be done in the future.

Computations were carried out for various situations and the results are given in §5.

5. Results of the computations and discussions.

Since there are five parameters (α, R, T_0, a, H_0) and the solution contains the free boundary $H(x)$, velocity $(u, v) = v_x, v_r)$, pressure p, and flux Q, computations were carried out by changing each of the parameters, and solutions were observed to study the properties of the flow. We have made the following observations.

1. *Accuracy of the long wave approximation and numerical method.* From Table 1 we can see that the numerical method is roughly of second-order accuracy. The long wave and numerical solutions agree with each other very well (Tables 2 and 3).

2. *Influence of T_0 on the flow.* (i) Phase shift of the max–min of the free boundary. From Fig. 2 we see the phase shift of the max–min of the free boundary when T_0 is not large. However, when T_0 becomes large, the phase shift becomes small, and eventually becomes zero.

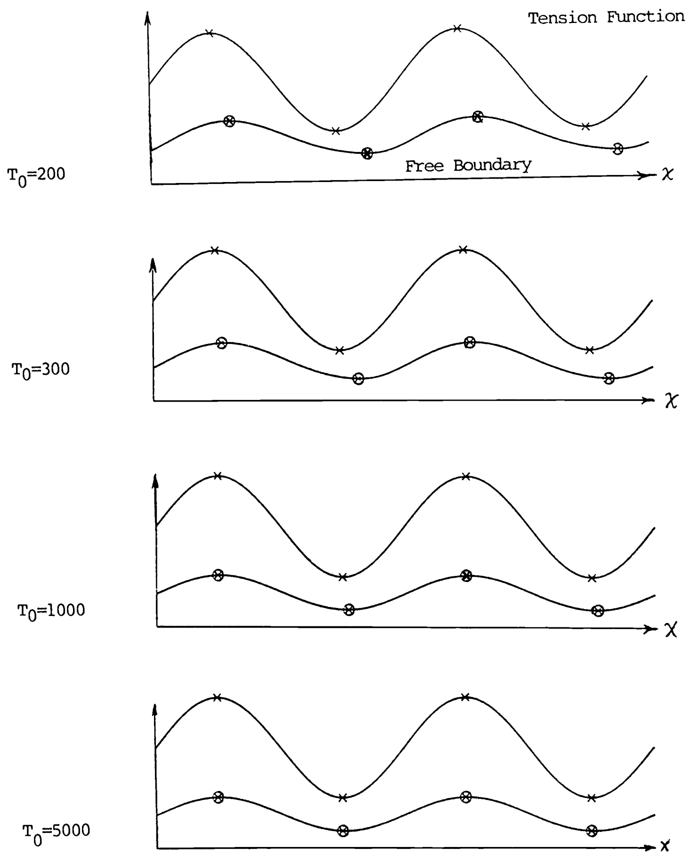


FIG. 2. *Free boundaries with T_0 changing.* $\alpha = 0.1$, $R = 0.1$, $H_0 = 0.5$, $a = 0.5$, $T_0 = 200$—5000.

(ii) Flow pattern. Our computation also indicates when T_0 becomes larger, the positive flow portion becomes larger and the tube becomes narrower.

(iii) $T_0 - Q$ curve. Figure 3 shows that the relation between T_0 and flux is not linear, and especially that the flux increases with T_0 very slowly when T_0 is greater than 2000.

(iv) Table 4 shows the max–min of the solutions with T_0 changing.

3. *Backflow and positive motion.* The v_x-minimum (negative) always appears at the narrower part of the tube, indicating that the fluid is leaking there. On the other hand, there are parts of "positive motion" at the wider part of the tube indicating the fluid is pushed forward by the wave. Usually the positive motion part is of a torus shape (note that the tube is axisymmetric).

4. *Pressure field.* Figure 4 gives the contour lines of the pressure fields. The picture shows the maximum of pressure appears at the right side of the "neck" and minimum

TABLE 1

Order of accuracy of the numerical method. $R = 0.2$, $T_0 = 250$, $a = 0.5$, $H_0 = 0.5$, $\ell = 5$, $d = 1$, $\alpha = 0.2$. *Numerical parameters:* $d_1 = \ell/m$, $d_2 = d/n$, $\omega = 1.5$, $\gamma = 0.1$, *local iteration* $= 20$, *main iteration* $= 200$.

Imbalance of equations by the numerical solutions							
m	n	Eq. (2.1)	Eq. (2.2)	Eq. (2.3)	$\|u\|$	$\|v\|$	$\|p\|$
10	6	0.000099	0.000114	0.114018	2.288	0.390	147
20	12	0.000021	0.000023	0.037850	2.023	0.356	131
40	24	0.000009	0.000012	0.023615	1.962	0.341	126

Notes: (1) *Imbalance of an equation by the numerical solution is defined in the text.* (2) *Equations* (2.1) *and* (2.2) *are equations of motion;* (2.3) *is the equation of continuity.*

TABLE 2

Comparison between the long wave and numerical solutions. Re $= 1.0 \times \alpha$, $T_0 = 10/a^2$, $a = 0.5$, $H_0 = .5$, $C = 1.00$.

	Imbalance of equations			L_2-norms			
α	Eq. (2.1)	Eq. (2.2)	Eq. (2.3)	u	v	p	H
$a = 0.2$							
long	.00366	.00000	.62605	2.110	.000	127.8	1.402
exact	.00051	.00027	.03053	1.978	.342	122.9	1.416
$a = 0.1$							
long	.00248	.00000	.44182	2.976	.000	714.2	1.975
exact	.00013	.00038	.03014	2.925	.254	713.6	1.978
$a = 0.05$							
long	.00223	.00000	.31195	4.203	.000	4014.8	2.787
exact	.00002	.00007	.00888	4.182	.177	4014.9	2.789
$a = 0.025$							
long	.00218	.00000	.22039	5.940	.000	22638.1	3.938
exact	.00000	.00001	.00362	5.932	.124	22638.5	3.938
$a = 0.0125$							
long	.00217	.00000	.15576	8.398	.000	127852.7	5.566
exact	.00000	.00000	.00147	8.395	.088	127853.4	5.566
$a = 0.00625$							
long	.00217	.00000	.11011	11.874	.000	722654.9	7.870
exact	.00000	.00000	.00298	11.873	.061	722655.9	7.870
$a = 0.003125$							
long	.00217	.00000	.07785	16.791	.000	4086283.8	11.128
exact	.00000	.00000	.00543	16.791	.043	4086285.0	11.128

TABLE 3

Relative errors between long wave and numerical solutions. Re $= 1.0 \times \alpha$, $T_0 = 10/a^2$, $a = 0.5$, $H_0 = 0.5$, $C = 100$.

	L_2-norms of relative errors			L_2-norms of exact solutions			
α	$(u - u0)/u$	$(p - p_0)/p$	$(H - H0)/H$	u	v	p	H
.2	.08034	.108214	.011228	1.98	.342	122.9	1.4020
.10	.02422	.010197	.002264	2.93	.254	713.6	1.9749
.05	.00730	.002346	.000662	4.18	.177	4014.9	2.7873
.025	.00211	.000595	.000177	5.94	.126	22854.7	3.9498
.0125	.00051	.000143	.000043	8.39	.088	127853.4	5.5662
.00625	.00013	.000033	.000010	11.87	.061	722655.9	7.8698
.003125	.00003	.000007	.000002	16.79	.043	4086285.0	11.1281

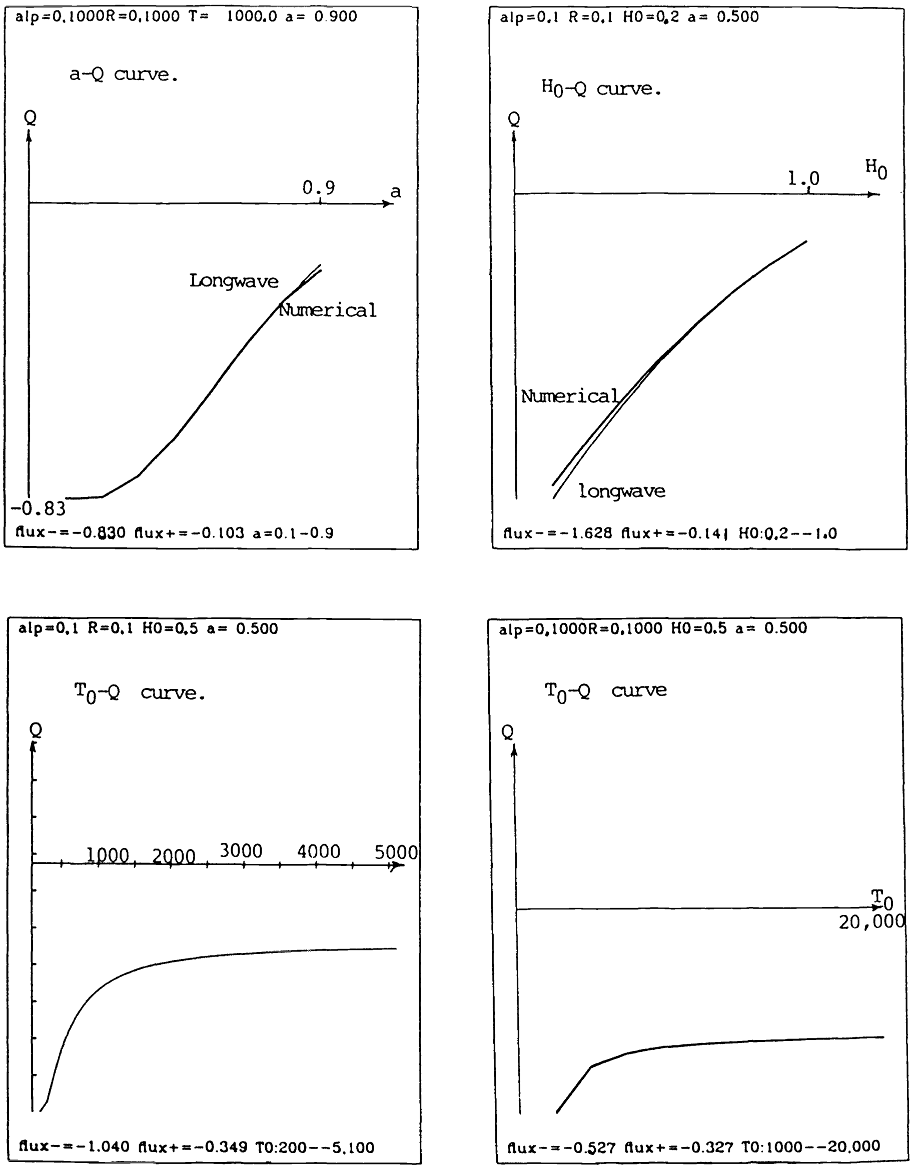


FIG. 3. *Relations between flux Q and other parameters.*

appears at the left side of the neck. So the pressure increases when passing the neck and decreases when going through the wider part of the tube. This agrees with the velocity pictures.

5. *Influence of Reynolds number on the flow.* Table 5 shows that the solution is not sensitive to the changes of Reynolds number when $0.001 \leq \text{Re} \leq 1000$ while $\text{Re} \cdot T_0 = 100$ is maintained. Numerically, smaller iterative parameters should be chosen to compute for large R values.

6. *Influence of a-change on the flow.* Figure 5 gives the velocity fields with respect

TABLE 4

Max–min *of solutions with* T_0 *changing.* $200 \leq T_0 \leq 1000$, $1000 \leq T_0 \leq 20{,}000$, $\alpha = 0.1$, $R = 0.1$, $a = 0.5$, $H_0 = 0.5$, $c = 1.0$.

T_0	$x$max–min		Hmin–max		Umin–Umax		Vmin–Vmax		Pmin–Pmax		flux
					Numerical solutions						
200	.400	.925	.493	.788	−1.69	0.65	−.113	0.89	179	469	−1.021
300	.350	.900	.468	.903	−1.87	.220	−.168	.128	278	628	−0.983
400	.325	.875	.437	.935	−1.97	.359	−.191	.145	395	806	−0.875
500	.325	.875	.407	.932	−2.03	.434	−.197	.152	529	1000	−0.774
600	.300	.850	.382	.920	−2.06	.481	−.198	.156	677	1201	−0.694
700	.300	.850	.363	.906	−2.09	.512	−.197	.158	833	1408	−0.633
800	.300	.850	.348	.892	−2.10	.534	−.195	.159	997	1614	−0.587
900	.300	.825	.337	.880	−2.12	.550	−.193	.159	1166	1825	−0.551
1000	.275	.825	.327	.870	−2.14	.564	−.191	.159	1340	2034	−0.522
2000	.275	.800	.283	.815	−2.19	.614	−.177	.159	3212	4094	−0.403
4000	.250	.775	.264	.783	−2.22	.625	−.167	.158	7143	8146	−0.356
6000	.250	.775	.259	.772	−2.21	.629	−.164	.157	11124	12163	−0.343
8000	.250	.775	.257	.767	−2.21	.630	−.162	.157	15116	16174	−0.337
10000	.250	.750	.256	.763	−2.22	.630	−.161	.157	19110	20180	−0.333
12000	.250	.750	.254	.761	−2.22	.631	−.160	.157	23106	24184	−0.331
14000	.250	.750	.254	.759	−2.22	.631	−.160	.157	27103	28187	−0.330
16000	.250	.750	.253	.758	−2.23	.631	−.159	.157	31101	32189	−0.328
18000	.250	.750	.253	.757	−2.23	.631	−.159	.157	35100	36190	−0.328
20000	.250	.750	.252	.757	−2.23	.631	−.159	.157	39098	40191	−0.327
					Long wave solutions						
200	.400	.950	.493	.798	−1.72	.041	.000	.000	177	462	−1.040
300	.350	.900	.466	.910	−1.92	.233	.000	.000	276	622	−0.997
400	.325	.875	.431	.934	−2.02	.357	.000	.000	396	803	−0.881
500	.325	.875	.401	.928	−2.08	.425	.000	.000	533	1000	−0.777
600	.300	.850	.377	.915	−2.12	.470	.000	.000	683	1202	−0.698
700	.300	.850	.359	.902	−2.15	.501	.000	.000	840	1409	−0.638
800	.300	.850	.345	.889	−2.16	.523	.000	.000	1004	1616	−0.592
900	.300	.825	.334	.877	−2.18	.540	.000	.000	1174	1826	−0.556
1000	.275	.825	.324	.867	−2.20	.553	.000	.000	1348	2034	−0.527
2000	.275	.800	.283	.814	−2.24	.609	.000	.000	3219	4092	−0.407
4000	.250	.775	.264	.782	−2.26	.628	.000	.000	7150	8141	−0.358
6000	.250	.775	.259	.772	−2.26	.633	.000	.000	11131	12157	−0.344
8000	.250	.750	.257	.766	−2.25	.634	.000	.000	15123	16168	−0.338
10000	.250	.750	.255	.763	−2.26	.635	.000	.000	19116	20174	−0.334
12000	.250	.750	.254	.761	−2.26	.635	.000	.000	23112	24178	−0.332
14000	.250	.750	.254	.759	−2.26	.636	.000	.000	27109	28180	−0.330
16000	.250	.750	.253	.758	−2.27	.636	.000	.000	31107	32183	−0.329
18000	.250	.750	.253	.757	−2.27	.636	.000	.000	35106	36184	−0.328
20000	.250	.750	.252	.757	−2.27	.646	.000	.000	39104	40185	−0.327

to the moving frame for $a = 0.2$ and 0.9. When a is small, there is no positive flow and no trapping. When a gradually increases, a small positive flow region appears near the center of the tube. When a becomes larger, the positive flow region becomes larger. a is the main parameter that has a major influence on flux, i.e., the efficiency of the fluid transport. The $a - Q$ curve is given in Fig. 3 for the a values between 0.1–0.9. The curves show that (i) when a is too small ($a \leq 0.2$), the flux is not sensitive to a change for the simple reason that the wave is not deep enough to push the fluid forward. (ii) When a is greater than 0.3, the flux increases almost linearly with a.

TABLE 5

Max-min *of solutions with R changing,* $0.001 \le \text{Re} \le 1000$. $\alpha = 0.1, R \cdot T_0 = 100, a = 0.3, H_0 = 0.5, C = 1.0$.

Numerical solutions												
R	$x$max–min		Hmin–max		Umin–Umax		Vmin–Vmax		Pmin	Pmax	Flux
0.001	.30	.825	.411	.719	−1.878	.040	−0.105	.090	155694	200000	−0.765
0.010	.30	.825	.411	.720	−1.872	.052	−0.105	.090	15557	20000	−0.765
0.100	.30	.825	.411	.719	−1.878	.040	−0.105	.090	1556	2000	−0.765
1.000	.30	.825	.411	.719	−1.877	.039	−0.105	.090	155	200	−0.765
10.000	.30	.825	.412	.722	−1.890	.050	−0.110	.094	15.4	20.0	−0.765
100.000	.30	.825	.411	.720	−1.902	.054	−0.104	.091	1.6	2.0	−0.765
1000.000	.30	.825	.411	.720	−1.902	.054	−0.104	.091	0.2	0.2	−0.765
Long wave solutions											
R	$x$max–min		Hmin–max		Umin–Umax		Vmin–Vmax		Pmin	Pmax	Flux
0.001	.30	.825	.410	.718	−1.901	.054	0.000	.000	156402	200000	−0.765
0.010	.30	.825	.410	.718	−1.901	.054	0.000	.000	15640	20000	−0.765
0.100	.30	.825	.410	.718	−1.901	.054	0.000	.000	1564	2000	−0.765
1.000	.30	.825	.410	.718	−1.901	.054	0.000	.000	156	200	−0.765
10.000	.30	.825	.410	.718	−1.901	.054	0.000	.000	15.6	20.0	−0.765
100.000	.30	.825	.410	.718	−1.901	.054	0.000	.000	1.6	2.0	−0.765
1000.000	.30	.825	.410	.718	−1.901	.054	0.000	.000	0.2	0.2	−0.765

Notes: Re *and* T_0 *are both changing while* Re·T_0 *remains constant. Smaller iterative parameters were used for greater R values.*

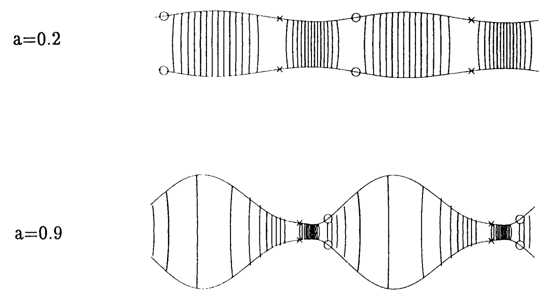


FIG. 4. *Pressure field with a changing.* $\alpha = 0.1$, $R = 0.1$, $H_0 = 0.5$, $T_0 = 1000$, $a = 0.2$—0.9.

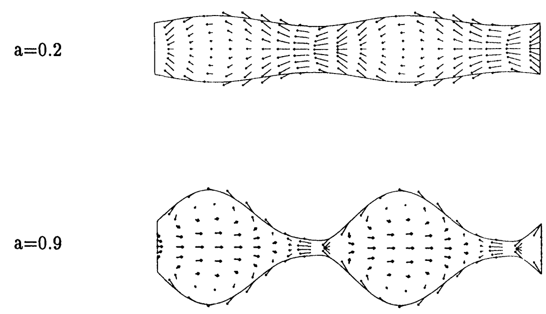


FIG. 5. *Velocity field of numerical solutions with a changing.* $\alpha = 0.1$, $R = 0.1$, $H_0 = 0.5$, $T_0 = 1000$, $a = 0.2$—0.9.

When drawing the velocity fields, the darker lines indicate forward motion while the lighter lines indicate backward motion. In the pressure field, circles indicate where the pressure is maximum and X indicates a minimum.

Acknowledgments. The authors wish to express their sincere appreciation to Professor C. S. Peskin and the referees for many helpful suggestions that improved the presentation of the paper.

REFERENCES

[1] A. R. BESTMAN, *Long wavelength peristaltic pumping in a heated tube at low Reynolds numbers*, Develop. in Mech., 10 (1979), pp. 195–199.

[2] K. DEIMLING, *Nonlinear Functional Analysis*, Springer-Verlag, New York, 1985.

[3] Y. C. FUNG AND C. S. YIH, *Peristaltic transport*, J. Appl. Mech., 37 (1968), pp. 901–905.

[4] Y. C. FUNG, *Biodynamics: Circulation*, Springer-Verlag, New York, 1984.

[5] ———, *Biomechanics: Motion, Flow, Stress, and Growth*, Springer-Verlag, New York, 1990.

[6] D. P. GIDDENS, C. K. ZARINS, AND S. GLAGOV, *Response of arteries to near-wall fluid dynamic behavior*, Appl. Mech. Rev., 43 (1990), pp. S98–S102.

[7] T. K. HUNG AND T. D. BROWN, *Solid particle motion in two dimensional peristaltic flows*, J. Fluid Mech., 73 (1976), pp. 77–96.

[8] M. Y. JAFFRIN AND A. H. SHAPIRO, *Peristaltic pumping*, Ann. Rev. Fluid Mech., 3 (1971), pp. 13–36.

[9] M. R. KAIMAL, *Peristaltic pumping of a Newtonian fluid with particles suspended in it at low Reynolds number under long wavelength approximations*, Trans. ASME, 45 (1978), pp. 32–36.

[10] D. N. KU, D. P. GIDDENS, C. K. ZARINS, AND S. GLAGOV, *Pulsatile flow and atherosclerosis in the human carotid bifurcation: positive correlation between plaque location and low and oscillating stress*, Arteriosclerosis, Vol. 5, May/June 1985.

[11] J. S. LEE, *The pressure-flow relationship in long-wave propagation in large arteries*, Proc. Symp. ASME Applied Mechanics Division, New York, 1966, pp. 96–120.

[12] B. B. LIEBER AND D. P. GIDDENS, *Post-stenotic core flow behavior in pulsatile flow and its effects on wall shear stress*, J. Biomechanics, 23 (1990), pp. 597–605.

[13] H. M. LIEBERSTEIN, *Mathematical Physiology*, Elsevier, New York, 1973.

[14] J. C. STRIKWERDA, *Finite difference methods for the Stokes and Navier-Stokes equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 56–58.

[15] J. C. STRIKWERDA AND Y. M. NAGEL, *A numerical method for the incompressible Navier-Stokes equations in three-dimensional cylindrical geometry*, J. Comput. Physics, 78 (1988), pp. 64–78.

[16] J. C. STRIKWERDA, B. A. WADE, AND K. P. BUBE, *Regularity estimates up to the boundary for elliptic systems of difference equations*, SIAM J. Numer. Anal., 27 (1990), pp. 292–322.

[17] S. TAKABATAKE AND K. AYUKAWA, *Numerical study of two-dimensional peristaltic flows*, J. Fluid Mech., 122 (1982), pp. 439–465.

[18] S. TAKABATAKE, K. AYUKAWA, AND A. MORI, *Peristaltic pumping in circular cylindrical tubes: A numerical study of fluid transport and it efficiency*, J. Fluid Mech., 193 (1988), pp. 267–283.

[19] D. TANG AND M. C. SHEN, *Peristaltic transport of a heat conducting fluid subject to Newton's cooling law at the boundary*, Internat. J. Engrg. Sci., 27 (1989), pp. 809–825.

[20] ———, *Numerical and asymptotic solutions for the peristaltic transport of a heat-conducting fluid*, ACTA Mechanica 83 (1990), pp. 93–102.

[21] H. WINET, *On the quantitative analysis of liquid flow in physiological tubes*, MRC Tech. Sum. Rep. No. 2456, University of Wisconsin–Madison, 1982.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1320–1338, November 1993

A PARALLEL ALGORITHM FOR REDUCING SYMMETRIC BANDED MATRICES TO TRIDIAGONAL FORM*

BRUNO LANG†

Abstract. An algorithm is presented for reducing symmetric banded matrices to tridiagonal form via Householder transformations. The algorithm is numerically stable and is well suited to parallel execution on distributed memory multiple instruction multiple data (MIMD) computers. Numerical experiments on the iPSC/860 hypercube show that the new method yields nearly full speedup if it is run on multiple processors. In addition, even on a single processor the new method usually will be several times faster than the corresponding EISPACK and LAPACK routines.

Key words. symmetric banded matrix, orthogonal reduction, eigenvalues, parallel, error analysis

AMS subject classifications. 65F15, 65W05, 65G05, 68C25

1. Introduction.

Reduction to tridiagonal form is a major step in eigenvalue computations for symmetric matrices. If the matrix is full, then usually Householder's or Givens' algorithms are the methods of choice [9], [16]. However, for banded matrices, which is the case to be considered in the present paper, these algorithms are not optimal as they do not fully exploit the band structure. Then each reduction step produces fill-in until eventually the matrix will be full, resulting in $O(n^3)$ time and $O(n^2)$ memory requirements.

A "chasing" algorithm by Schwarz [15], [14] avoids this problem by removing *any* fill-in as soon as it occurs. This algorithm is implemented in the routine BANDR of the EISPACK library [8]. The algorithm relies on (fast) Givens rotations to selectively introduce zeroes into the band and to remove intermediate fill-in. The main part of the computations *could* be done with the _AXPY operation of the level-1 BLAS (basic linear algebra subprograms) [5], [6], [13]. However, as the EISPACK library dates back farther than the evolution of the BLAS, the routine BANDR contains no call to the latter.

Another version of the same underlying method is due to Kaufman [10]; by rearranging the computations she was able to increase the vector lengths, if the bandwidth d is very small compared to the matrix dimension n. This algorithm is incorporated into the routines _SBTRD of the recent LAPACK library [1]. Again, rotations are used to transform the matrix, but now in the form of ordinary Givens' rotations. This leads to an increased overall floating point operation (flop) count of $6n^2d$ versus $4n^2d$ flops of the BANDR version. Another drawback of this method lies in the fact that the bulk of computation is coded in explicit loops since there are no appropriate BLAS routines to cover them.

Both algorithms make full use of the symmetry in the given matrix and require only the lower (or upper) half of the band to be stored. Thus, $(d+1)n$ memory locations are needed.

In this paper we introduce a new reduction algorithm SBTH (symmetric banded matrix, reduction to tridiagonal form via Householder transformations) which is based—as the name indicates—on Householder transformations. The flop count for this algorithm is roughly the same as for the _SBTRD routines from LAPACK and therefore about 50% higher than that of the EISPACK routine.

*Received by the editors June 15, 1992; accepted for publication (in revised form) November 12, 1992. The numerical experiments were carried out on the 8 node and 128 node iPSC/860 hypercubes of the Advanced Computing Research Facility at Argonne National Laboratory and of Oak Ridge National Laboratory, respectively.

†Fachbereich 7—Mathematik, Bergische Universität GH Wuppertal, Gauss-Str. 20, W-5600 Wuppertal 1, Germany.

The new method has some distinct advantages over the above-mentioned implementations. First, almost all the computations can be done within the level-2 BLAS; thus, SBTH benefits from a considerably higher *data locality* than the other algorithms, a significant bonus on machines with low bandwidth from the main memory to the (vector) arithmetical unit(s). In fact, if all three algorithms are run on *one* processor of the iPSC/860 hypercube parallel computer, the new method may perform several times better than the other algorithms. In addition, the new algorithm is well suited to parallel execution. Theoretical considerations as well as numerical experiments show that execution of this algorithm on multiple processors will yield nearly full speedup, provided that the matrix is "reasonably large." The only major drawback of the new algorithm is its higher storage requirements ($2dn$ memory locations).

The remainder of the paper is organized as follows. Section 2 introduces a block decomposition of the band matrix and a serial version of our new reduction algorithm SBTH, which is based on this decomposition. In §3 we show how to efficiently parallelize this algorithm by mapping the blocks to different processors. An additional level of parallelism can be obtained by also performing the transformation of each block in parallel, as explained in §4. The reader may be warned that, in order to achieve optimal performance, the algorithms will become increasingly tricky. But the additional complexity is paying well, as will be demonstrated by performance data from experiments on the iPSC/860 hypercube (§5). Finally, §6 concludes with the summary of a detailed error analysis and with comments on another variant of the reduction scheme.

2. The reduction process.

2. The reduction process. Throughout the paper, $A = (a_{ij})$ always denotes a symmetric banded matrix of dimension n and (semi)bandwidth d, i.e., $a_{ij} = 0$ for $|i-j| > d$.

The reduction process consists of applying $n-2$ orthogonal transformations

$$A^{(\nu+1)} := Q^{(\nu)^T} \cdot A^{(\nu)} \cdot Q^{(\nu)}, \qquad \nu = 1, \ldots, n-2, \tag{1}$$

to the matrix $A^{(1)} := A$. The νth of these steps eliminates the νth column (and row) of the band, i.e., the elements $a_{\nu+2,\nu}, \ldots, a_{\nu+d,\nu}$ are made zero. Since all the $A^{(\nu)}$ are symmetric, only their lower triangle will be stored and used during the reduction.

Suppose that for the νth step we are able to partition $A^{(\nu)}$ as follows.

$$A^{(\nu)} = \left(\begin{array}{ccccc} T_\nu & & & & 0 \\ & A_{10}^{(\nu)T} & & & \\ A_{10}^{(\nu)} & A_{11}^{(\nu)} & A_{21}^{(\nu)T} & & \\ & A_{21}^{(\nu)} & A_{22}^{(\nu)} & \ddots & \\ 0 & & \ddots & \ddots & A_{b,b-1}^{(\nu)T} \\ & & & A_{b,b-1}^{(\nu)} & A_{bb}^{(\nu)} \end{array}\right)$$

The first $\nu - 1$ columns (and rows) of the band have already been removed; thus, the leading $\nu \times \nu$ submatrix T_ν of $A^{(\nu)}$ is tridiagonal. Let $n - \nu = (b-1) \cdot d + r$, $1 \le r \le d$. Then the blocks $A_{\beta\beta}^{(\nu)} \in \mathbb{R}^{d \times d}$ $(1 \le \beta < b)$ and $A_{bb}^{(\nu)} \in \mathbb{R}^{r \times r}$ along the diagonal are symmetric. At the beginning of the algorithm ($\nu = 1$), the subdiagonal blocks $A_{\beta+1,\beta}^{(\nu)} \in \mathbb{R}^{d \times d}$ $(1 \le \beta < b-1)$ are upper triangular, and $A_{b,b-1}^{(\nu)} \in \mathbb{R}^{r \times d}$ is upper trapezoidal. For $\nu > 1$, however, these subdiagonal blocks will be full. $A_{10}^{(\nu)}$ denotes the first of the remaining columns of the band (except for its first element, which is contained in T_ν).

The corresponding partition of $Q^{(\nu)}$ is

$$Q^{(\nu)} = \begin{pmatrix} \boxed{I_\nu} & & & & 0 \\ & \boxed{Q_1^{(\nu)}} & & & \\ & & \boxed{Q_2^{(\nu)}} & & \\ 0 & & & \ddots & \\ & & & & \boxed{Q_b^{(\nu)}} \end{pmatrix}$$

with a $\nu \times \nu$ identity matrix I_ν and "suitable" orthogonal blocks $Q_\beta^{(\nu)} \in \mathbb{R}^{d \times d}$ $(1 \le \beta < b)$ and $Q_b^{(\nu)} \in \mathbb{R}^{r \times r}$.

Then the transformed matrix $\tilde{A} := Q^{(\nu)^T} \cdot A^{(\nu)} \cdot Q^{(\nu)}$ has a block structure compatible with that of $A^{(\nu)}$. For its blocks we obtain the relations

$$\begin{aligned} \tilde{A}_{10} &= Q_1^{(\nu)^T} \cdot A_{10}^{(\nu)}, \\ \tilde{A}_{\beta\beta} &= Q_\beta^{(\nu)^T} \cdot A_{\beta\beta}^{(\nu)} \cdot Q_\beta^{(\nu)} \quad (1 \le \beta \le b), \\ \tilde{A}_{\beta+1,\beta} &= Q_{\beta+1}^{(\nu)^T} \cdot A_{\beta+1,\beta}^{(\nu)} \cdot Q_\beta^{(\nu)} \quad (1 \le \beta < b). \end{aligned}$$

Remember that, by (1), we also should have $\tilde{A} = A^{(\nu+1)}$. The block decomposition of the matrix $A^{(\nu+1)}$, however, is shifted one column to the right and one row to the bottom as compared to the decomposition of $A^{(\nu)}$ and $\tilde{A}$. This implies that some of the elements of $\tilde{A}$ will no longer be included in the blocks of $A^{(\nu+1)}$. To be more specific, the elements at positions $(2,1), \ldots, (d,1)$ in the first column of each subdiagonal block $\tilde{A}_{\beta+1,\beta}^{(\nu)}$ $(0 \le \beta < b)$ will be lost when the block decomposition is shifted. Therefore, we must make sure that they *are already zero*; this can be enforced by a suitable choice of the transformation matrices $Q_{\beta+1}^{(\nu)}$ (see below).

This leads to the following serial reduction algorithm (for ease of reading, the superscripts $^{(\nu)}$ have been dropped):

ALGORITHM 2.1. *Serial reduction algorithm, based on Householder transformations.*

```
for $\nu = 1$ to $n-2$
    (adopt the block decomposition for step $\nu$)
    determine $b$ and $r$ such that $n - \nu = (b-1) \cdot d + r$ with $1 \leq r \leq d$
    determine an orthogonal matrix $Q_1$ such that $\tilde{A}_{10} := Q_1^T \cdot A_{10}$ has the
        form $(*, 0, \ldots, 0)^T$; replace $A_{10}$ by $\tilde{A}_{10}$
    for $\beta = 1$ to $b$
        replace $A_{\beta\beta}$ by $\tilde{A}_{\beta\beta} := Q_\beta^T \cdot A_{\beta\beta} \cdot Q_\beta$
        if $\beta < b$
            determine an orthogonal matrix $Q_{\beta+1}$ such that the first column
                of $\tilde{A}_{\beta+1,\beta} := Q_{\beta+1}^T \cdot (A_{\beta+1,\beta} \cdot Q_\beta)$ has the form $(*, 0, \ldots, 0)^T$;
            replace $A_{\beta+1,\beta}$ by $\tilde{A}_{\beta+1,\beta}$
```

As indicated earlier, the blocks Q_β may be chosen as Householder matrices of size $d \times d$ or—in the case of Q_b—$r \times r$. Q_1 eliminates all but the first elements of A_{10}, which is the first of the remaining columns in the band. To determine $Q_{\beta+1}$ $(\beta > 0)$, one computes the first column of the matrix $A_{\beta+1,\beta} \cdot Q_\beta$ and then choses $Q_{\beta+1}$ to eliminate all but the first elements of this column d-vector.

3. Parallelism between different blocks. The serial algorithm bears a twofold potential for exploiting parallelism. First, *different* blocks may be transformed simultaneously. In this section we will use a modified pipelined approach to make use of this medium-grained *parallelism between different blocks*. An additional level of parallelism can be obtained by further distributing the work for the transformation of *single* blocks to multiple processors. This—slightly finer-grained—*parallelism within the blocks* will be treated in §4.

In the sequel, we will always treat the blocks by pairs; to make this explicit we introduce the notation of a *block pair*:

$$B_\beta := \begin{pmatrix} A_{\beta\beta} \\ A_{\beta+1,\beta} \end{pmatrix} \quad (\beta \geq 0).$$

To keep the parallel algorithms concise we need some more abbreviations.

$$\text{“}\textit{eliminate } B_0 \rightarrow Q_1\text{”}$$

stands for the fourth and fifth line of Algorithm 2.1 (“*determine* ... $\tilde{A}_{10}$”),

$$\text{“}Q_\beta \rightarrow \textit{transform } B_\beta \rightarrow Q_{\beta+1}\text{”}$$

comprises lines 7 through 10 of the algorithm. These abbreviations emphasize another important point: the new transformation matrix $Q_{\beta+1}$ is determined during the transformation of B_β (more precisely, of $A_{\beta+1,\beta}$).

There are three major steps in changing the serial algorithm into an efficient parallel program that makes best use of the parallelism between different blocks. First, we will make it a *distributed* algorithm. This means that the work is distributed among the processors, but that at any given time only one processor is active. Then, a “local view” of the block decomposition introduced in §2 will be the key for introducing parallelism. Last, we will optimize the data layout and the algorithmic flow to improve the performance.

We begin by distributing the block pairs to different *clusters* C_β of processors, $\beta = 0, 1, \ldots$, such that block pair B_β is assigned to cluster C_β (cf. the first picture of Fig. 1).

Here, a cluster consists of $p \geq 1$ processors. At the moment, it is sufficient to think of the clusters as of single processors (i.e., $p = 1$) as the value of p has no direct bearing on the algorithms in this section. (The use of multiple processors per cluster will be explained in §4, when we discuss the implementation of the block transformations in some detail.) We assume that the clusters are linearly connected to form a chain.

To match the data distribution, the work of Algorithm 2.1 must also be distributed among the clusters. In each pass through the ν loop, cluster C_0 begins by eliminating the first of the remaining columns of the band. Then C_0 sends the transformation matrix $Q_1^{(\nu)}$ to C_1 which transforms its block pair B_1 and thereby generates a new transformation matrix $Q_2^{(\nu)}$. Now C_1 passes this matrix to its successor C_2 which in turn transforms its block pair B_2 and determines $Q_3^{(\nu)}$. Continuing in this way, the activity moves along the chain of clusters until all the block pairs are processed. At this moment, the block decomposition must be shifted for the next pass through the ν loop. This implies that—except for C_0—each cluster C_β must send the first column of its transformed block pair to its predecessor $C_{\beta-1}$.

A simple modification of this scheme allows some of the clusters to operate simultaneously. The key observation is that the first column of the transformed B_1 is not affected by the ensuing operations in any block pair $B_\beta, \beta > 1$. Therefore, C_1 can send this column back to C_0 as soon as the transformation of B_1 is completed; there is no need to wait until *all* the computations corresponding to $Q^{(\nu)}$ are done. This enables C_0 to *locally* shift its block pair B_0 and to eliminate the $(\nu + 1)$st column of the band, while C_2 is still transforming its block pair according to $Q_2^{(\nu)}$. The same argument also applies to the other clusters C_β. As soon as they receive a column from their successor $C_{\beta+1}$ they locally shift their own block pair and apply the next transformation to it.

Figure 1 gives a sketch of the "start up" of the resulting parallel Algorithm 3.1. To let the complete tridiagonal matrix T assemble in C_0 the ν loop ends with $n - 1$ instead of $n - 2$, as it did in Algorithm 2.1. The additional last pass requires no computation. Of course, we do not send the Householder matrices as such, but only the associated Householder vectors.

ALGORITHM 3.1. *Parallel reduction algorithm, raw version (two different programs for cluster* 0 *and for the remaining clusters, respectively).*

```
$C_0$   :   for $\nu = 1$ to $n - 1$
              eliminate $B_0 \to Q_1$
              send $(Q_1)$ to $C_1$
              receive $(B_0)$ from $C_1$

$C_\beta$ $(\beta \geq 1)$ for $\nu = 1$ to $n - 1$
              if $B_\beta$ is not empty
                 receive $(Q_\beta)$ from $C_{\beta-1}$
                 $Q_\beta \to$ transform $B_\beta \to Q_{\beta+1}$
                 send (first column of $B_\beta$) to $C_{\beta-1}$
                 shift $B_\beta$ to the right/bottom by one column/row
                 if $B_{\beta+1}$ is not empty
                    send $(Q_{\beta+1})$ to $C_{\beta+1}$
                    receive (last column of $B_\beta$) from $C_{\beta+1}$
```

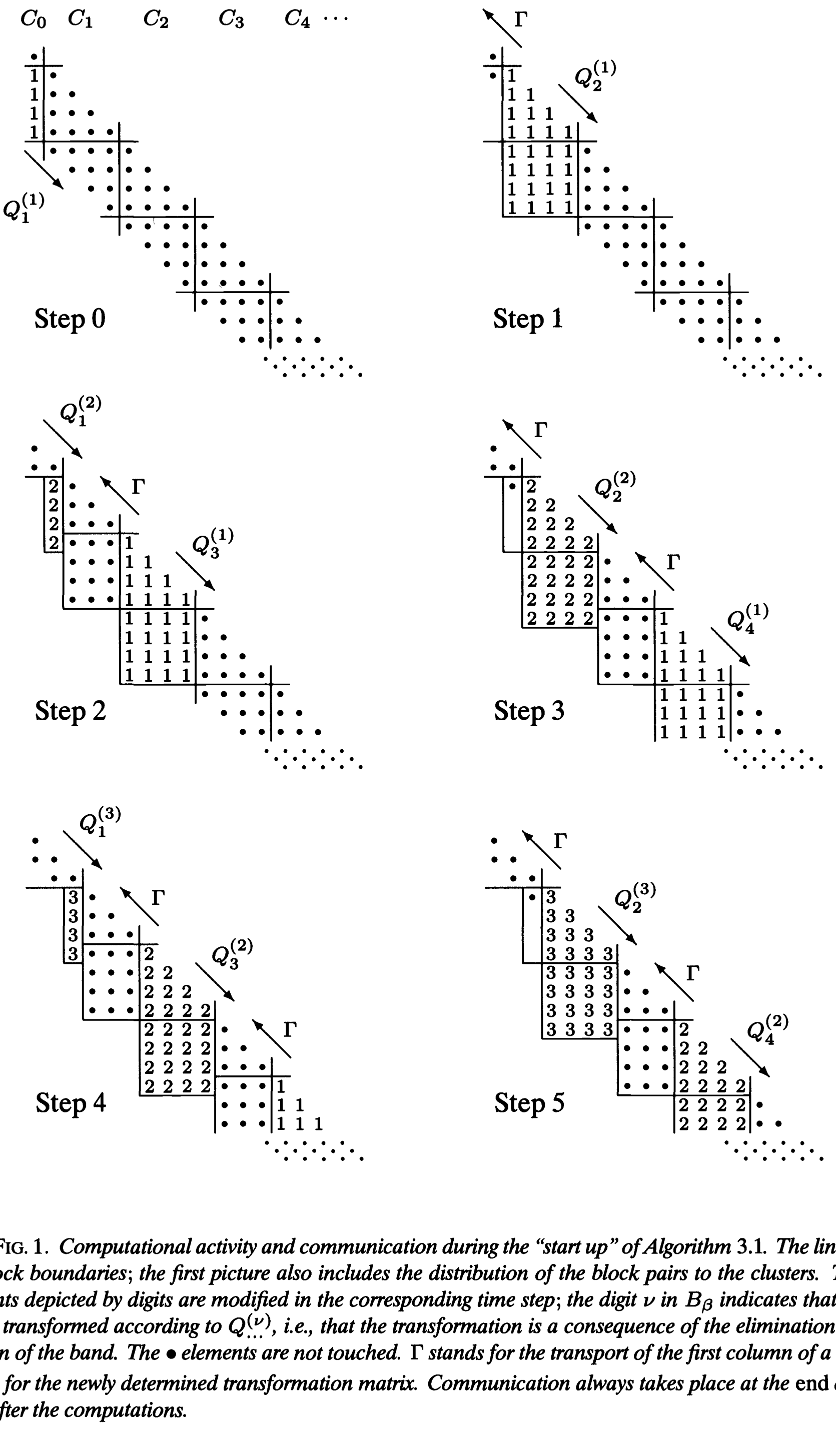


FIG. 1. *Computational activity and communication during the "start up" of Algorithm* 3.1. *The lines indicate the block boundaries; the first picture also includes the distribution of the block pairs to the clusters. The matrix elements depicted by digits are modified in the corresponding time step; the digit ν in B_β indicates that this block pair is transformed according to $Q_{\dots}^{(\nu)}$, i.e., that the transformation is a consequence of the elimination of the νth column of the band. The • elements are not touched. Γ stands for the transport of the first column of a block pair, $Q_{\beta+1}^{(\nu)}$ for the newly determined transformation matrix. Communication always takes place at the* end *of the time step, after the computations.*

As the transformations according to $Q^{(1)}$ progress along the band, more and more clusters become active (cf. Fig. 1), until the end of the band is reached. Later on, as the remaining band becomes shorter, an increasing number of clusters fall back to inactivity because their block pairs are "empty." It is not hard to see [12] that on the average, of the $c^* = \lceil (n-1)/d \rceil + 1$ clusters that shared the band at the start of the algorithm, only one half will hold a nonempty block pair. Therefore, one-half of the available computational power is lost by load imbalance.

In addition, only one-half of the clusters with nonempty block pairs can work at any given time, as may be observed from Fig. 1. The other clusters are idle because they are waiting for transformation matrices from their left neighbors and for columns from their right neighbors. Thus, the attainable speedup is reduced by another factor of 2.

Fortunately, there is a simple way to circumvent much of this loss in efficiency if we are willing to use less than the c^* clusters that were assumed for Algorithm 3.1. By halving the number of clusters we can completely eliminate the "idle waiting" time mentioned above, and further reduction of the number of clusters (to $n/(10d)$, say) almost eliminates the load imbalance resulting from empty block pairs. (In §4 we will show that this does not preclude utilizing a much larger number of processors.)

The "idle waiting" time is a result of the fact that adjacent block pairs cannot be transformed at the same time. Therefore, we now do not distribute the band to the clusters by block pairs but by *logical blocks* L_λ $(\lambda \geq 0)$, where a logical block consists of $l \geq 2$ consecutive block pairs. l is called the *blocking factor*. (L_0 is an exception; it contains only the first column of the band, that is, B_0.) In addition, we use a cyclic (wrapped) mapping: if c clusters $C_1, \ldots, C_c$ are available, then C_γ obtains the logical blocks L_λ with $\lambda \equiv \gamma \bmod c$. This requires the clusters to be connected as a ring. Note that now C_c holds the first column B_0 of the band and therefore takes on the work of cluster C_0 from Algorithm 3.1.

Now, in each pass through the ν loop of the algorithm, a cluster generally will have to process more than one logical block, and for each logical block there are l consecutive block pairs to transform. To this end we subdivide the body of the ν loop into l *phases* $\varphi = 1, \ldots, l$. Before the first phase starts, the cluster receives b transformation matrices from its left neighbor. (b varies with the progress of the algorithm. It starts with $b = 0$ as long as the activity has not yet reached the cluster, then slowly increases to the number of logical blocks that are stored in the cluster, and finally decreases with the number of nonempty logical blocks.) During the first phase the cluster transforms the first block pair of (the first b of) its logical blocks and determines the new transformation matrices. Between the first and the second phase, the first columns of the transformed block pairs are sent back to the left neighbor where they are stored as the last columns of the corresponding lth block pairs. In the phases $\varphi = 2, \ldots, l$ the cluster transforms the φth block pair in each logical block, using the most recently determined transformation matrices and producing new ones. Only after the lth phase, further communication is needed, as the transformation matrices must be passed to the right neighbor.

A more formal description of the above is given in Algorithm 3.2. To save space, the selection "**if** *me* $= C_c$ ..." is condensed to "[...]"; thus, statements in square brackets are executed only by cluster C_c. To keep the indices readable, we have used a "local" numbering of the block pairs: $B_{(\gamma:\beta:\varphi)}$ is the φth block pair in the βth logical block of cluster C_γ (that is, the block pair $(\beta-1)lc+(\gamma-1)l+\varphi$ of the band). The transformation matrices are indexed correspondingly.

Figure 2 sketches the "start up" of this algorithm.

ALGORITHM 3.2. *Parallel reduction algorithm, optimized version (one program for all clusters $C_\gamma, \gamma = 1, \dots, c$).*

```
$b := 0$                          (* number of "active" (transformable) block pairs *)
[ eliminate $B_0 \to Q_1$ ]
for $\nu = 2$ to $n$
  send $([Q_1,]\ Q_{(\gamma:\beta:l+1)}\ :\ 1 \le \beta \le b)$ to $C_{\gamma+1}$
  receive $(Q_{(\gamma:\beta:1)}\ :\ 1 \le \beta \le b)$ from $C_{\gamma-1}$
          (* phase 1 *)
  for $\beta = 1$ to $b$
    $Q_{(\gamma:\beta:1)} \to$ transform $B_{(\gamma:\beta:1)} \to Q_{(\gamma:\beta:2)}$
    shift $B_{(\gamma:\beta:1)}$ by one column/row
  send (first column of $B_{(\gamma:\beta:1)}\ :\ 1 \le \beta \le b$) to $C_{\gamma-1}$
  receive $([B_0,]$ last column of $B_{(\gamma:\beta:l)}\ :\ 1 \le \beta \le b')$ from $C_{\gamma+1}$
          (* remaining phases *)
  for $\varphi = 2$ to $l$
    if $b > 0$ and $B_{(\gamma:b:\varphi)}$ is empty
      $b := b - 1$              (* my last logical block is not full *)
    for $\beta = 1$ to $b$
      $Q_{(\gamma:\beta:\varphi)} \to$ transform $B_{(\gamma:\beta:\varphi)} \to Q_{(\gamma:\beta:\varphi+1)}$
      shift $B_{(\gamma:\beta:\varphi)}$ by one column/row
  [ eliminate $B_0 \to Q_1$ ]
          (* for the next pass through the $\nu$ loop: *)
  if $b > 0$ and $B_{(\gamma+1:b:1)}$ is empty
    $b := b - 1$            (* the last block pair of $C_\gamma$ has no successor in $C_{\gamma+1}$ *)
```

In Algorithm 3.2 the integers b and b' are set by the **receive** operation. Thus, the algorithm is "self-regulating" in the sense that the number of logical blocks to transform and the number of block pairs to complete by a new column are automatically determined from the number of items received.

The communication pattern created by this algorithm is very regular, as indicated in Fig. 2. For each pass through the ν loop there is a cyclic shift to the left after phase 1 (columns) and a cyclic shift to the right after phase l (transformation matrices).

As compared to Algorithm 3.1, the overall volume of communication is reduced by a factor of l because only one out of every l transformation matrices and columns must be passed between different clusters. This might suggest that large values of l are desirable.

On the other hand, the loss of speedup, which results from the band getting shorter, is also reduced from a factor of 2 to approximately $1 + 1/\omega$, where ω is the initial *wrap* (the number of logical blocks assigned to each cluster at the beginning of the algorithm) [12, Lemma 5.2]. Because of $\omega \approx n/(cld)$ this implies that while a large value of l decreases communication volume, it also reduces the wrap and so increases the load imbalance between the clusters. In all our experiments $l = 2$ turned out to be optimal.

As mentioned above, the loss of efficiency due to the shrinking band is reduced from a factor of 2 to about $1 + 1/\omega$. This is a benefit of multiply wrapping the logical blocks onto the cluster ring. In addition, the grouping of the block pairs into logical blocks completely removes the idle waiting time and therefore prevents losing the second factor of 2 in efficiency. Thus, if we run the algorithm on c clusters and neglect the communication costs, we may expect a speedup of nearly $c/(1+1/\omega)$ over the one-cluster version. Taking a few percent (5, say) off this estimate to account for the overhead of parallel execution gives a very good prediction for the speedup attainable in practice (see §5).

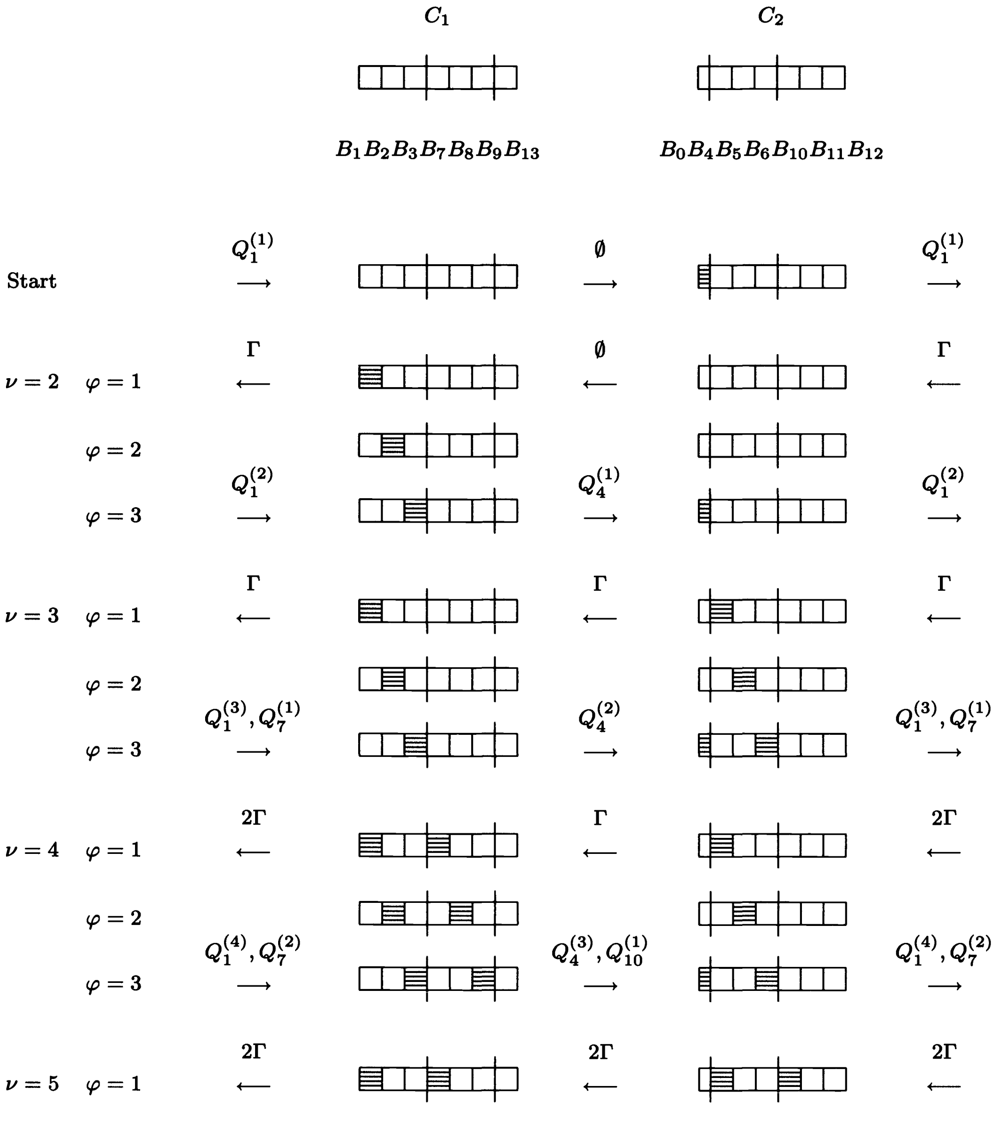


FIG. 2. *Distribution of the block pairs (for $c = 2$ clusters and blocking factor $l = 3$) and "start up" of Algorithm* 3.2. *The shaded block pairs are modified in the respective phase, the others remain unchanged. The Q's indicate communication of transformation matrices,* Γ *and* 2Γ *stand for the transport of one and two columns, respectively, and* $\emptyset$ *denotes empty communication. The communication is cyclic (indicated by the duplicated arrows and labels in front of the block pairs of cluster* C_1 *and behind those of* C_2*, respectively) and always takes place at the* end *of the phase, after the computations.*

4. Parallelism within each block. In this section we will focus on the implementation of the block operation "$Q_\beta \rightarrow$ *transform* $B_\beta \rightarrow Q_{\beta+1}$."

Let $Q_\beta = I - vv^T$ and $Q_{\beta+1} = I - \tilde{v}\tilde{v}^T$ with $||v|| = ||\tilde{v}|| = \sqrt{2}$. Then the computations for the transformation of the two blocks may be arranged as follows:

$$Q_\beta^T A_{\beta\beta} Q_\beta = A_{\beta\beta} - vw^T - wv^T$$

with

$$w := x - \frac{1}{2}(v^T x) \cdot v, \quad \text{where} \quad x := A_{\beta\beta} \cdot v,$$

cf. [7]. Analogously, we have

$$Q_{\beta+1}^T A_{\beta+1,\beta} Q_\beta = A_{\beta+1,\beta} - \tilde{x} v^T - \tilde{v} \tilde{w}^T$$

with

$$\tilde{x} := A_{\beta+1,\beta} \cdot v \quad \text{and} \quad \tilde{w} := \tilde{z} - (\tilde{v}^T \tilde{x}) \cdot v, \quad \text{where} \quad \tilde{z}^T := \tilde{v}^T \cdot A_{\beta+1,\beta}.$$

If the cluster consists of just one processor, then the following serial algorithm may be used to transform B_β. ($A_{\beta+1,\beta}(1 : d, 1)$ denotes the first column of that block.)

ALGORITHM 4.1. *"$I - vv^T = Q_\beta \rightarrow$ transform $B_\beta \rightarrow Q_{\beta+1} = I - \tilde{v}\tilde{v}^T$," serial algorithm.*

```
$x := A_{\beta\beta} \cdot v$
$\tilde{x} := A_{\beta+1,\beta} \cdot v$
        (* new Householder vector *)
$h := A_{\beta+1,\beta}(1 : d, 1) - v_1 \cdot \tilde{x}$        (* first column of $A_{\beta+1,\beta} Q_\beta$ *)
determine a Householder vector $\tilde{v}$, $||\tilde{v}|| = \sqrt{2}$, such that $(I - \tilde{v}\tilde{v}^T) \cdot h = (*, 0, \ldots, 0)^T$
        (* auxiliary vectors *)
$\tilde{z}^T := \tilde{v}^T \cdot A_{\beta+1,\beta}$
$w := x - \frac{1}{2}(v^T x) \cdot v, \quad \tilde{w} := \tilde{z} - (\tilde{v}^T \tilde{x}) \cdot v$
        (* rank-2 modifications *)
$A_{\beta\beta} := A_{\beta\beta} - vw^T - wv^T$
$A_{\beta+1,\beta} := A_{\beta+1,\beta} - \tilde{x}v^T - \tilde{v}\tilde{w}^T$
```

Note that the bulk of work can be done with level-2 BLAS. The computation of x, $\tilde{x}$, and $\tilde{z}^T$ requires a symmetric matrix vector product (remember that $A_{\beta\beta}$ is symmetric with only the lower triangle stored) and two general matrix vector products. The update of $A_{\beta\beta}$ is a symmetric rank-2 modification, whereas the update of $A_{\beta+1,\beta}$ may be realized by two subsequent general rank-1 modifications. These six calls to level-2 BLAS routines account for approximately $12d^2$ flops. The remaining $O(d)$ flops are covered by the level-1 BLAS.

If the number of available processors is not too high, then the "only parallelism between different blocks" version of our parallel algorithm performs best. The need for also exploiting parallelism within the transformation of single block pairs arises if the number of processors exceeds $c' = n/(ld)$. Since the band contains only c' logical blocks, adding more than c' clusters cannot further increase the speedup. Therefore, the only way to make use of a higher number of processors is to group multiple processors into a cluster and to distribute the work for one transformation among the processors of the cluster.

In the following discussion we assume that each cluster consists of $p \geq 1$ processors P_π, $\pi = 1, \ldots, p$, and that $2p \leq d$. Then we subdivide the block pair into $2p$ *block columns* of $\lfloor d/(2p) \rfloor$ or $\lceil d/(2p) \rceil$ consecutive columns each. Two block columns $\pm\pi$ are assigned to processor P_π, as indicated in Fig. 3. Thus, the first p of the block columns are distributed to the processors in reverse order, the other p block columns are distributed in their natural order. The boundary between $A_{\beta\beta}$ and $A_{\beta+1,\beta}$ further subdivides each block column σ into two *subblocks* $S_{1\sigma}$ and $S_{2\sigma}$; see Fig. 3.

A formal description of how the processors work together to transform the block pair is given in Algorithm 4.2. To simplify the notation, we partition the vectors $v, x, w, \tilde{w}$, and

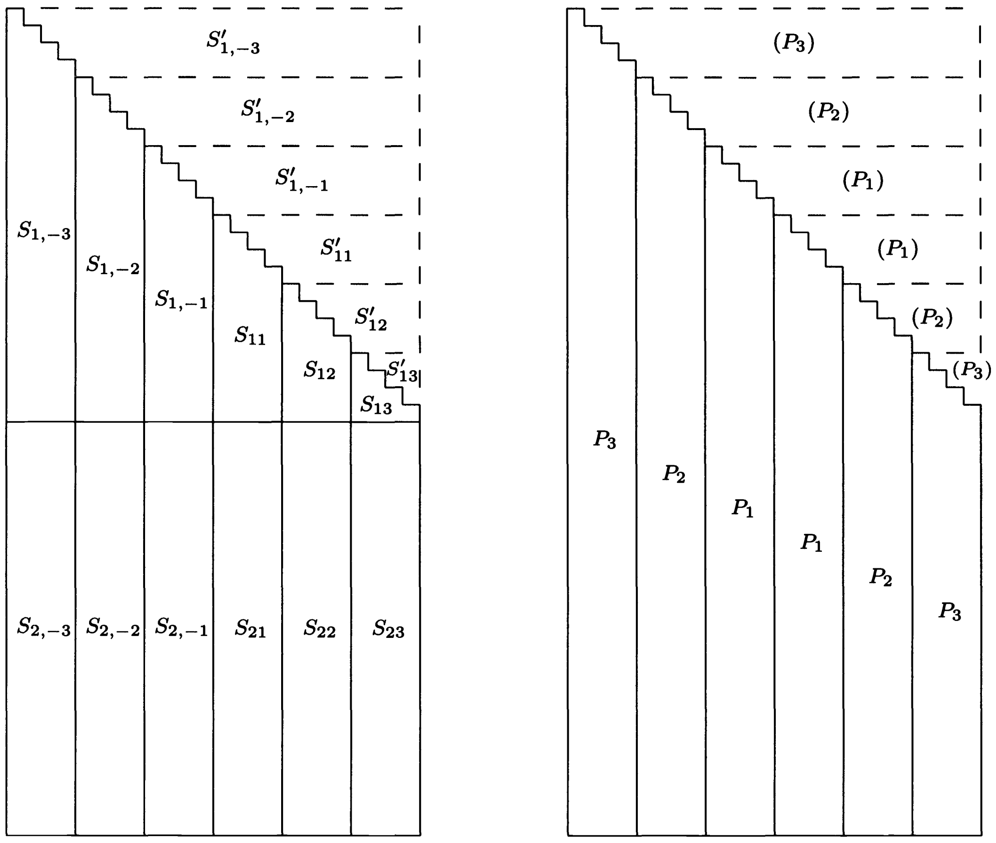


FIG. 3. *Distribution of the block pair to the processors of the cluster (second picture) and numbering of the subblocks for (semi)bandwidth $d = 24$ and $p = 3$ processors. The strict upper triangle of the block $A_{\beta\beta}$ is not explicitly stored; its entries are only indirectly accessible by transposing the lower triangle.*

$\tilde{z}$ to match the decomposition of B_β into block columns, e.g., $v^T = ({v^{(-p)}}^T, \ldots, {v^{(-1)}}^T, {v^{(1)}}^T, \ldots, {v^{(p)}}^T)$.

The computation of the matrix vector products $x = A_{\beta\beta} \cdot v$ and $\tilde{x} = A_{\beta+1,\beta} \cdot v$ is based on a decomposition of the sums involved. For example, we have

$$\tilde{x} = \sum_{\pi=\pm p,\ldots,\pm 1} S_{2\pi} \cdot v^{(\pi)}.$$

First, each processor P_π forms its "local contribution" $S_{2,-\pi} \cdot v^{(-\pi)} + S_{2\pi} \cdot v^{(\pi)}$ to the sum, and then these contributions are added across the processors to form the entire product $\tilde{x}$ in processor P_p. This final summation may be done with a **fan-in**-type operation. The computation of x is similar, with a slight complication. Here one must account for the upper triangle of $A_{\beta\beta}$; these elements are accessible by transposing the subblocks of the lower triangle [11]. When x and $\tilde{x}$ are collected in P_p, this processor—which also holds the first column of $A_{\beta+1,\beta}$—determines the new Householder vector $\tilde{v}$. As all the processors need the vectors x, $\tilde{x}$, and $\tilde{v}$ to proceed, these are broadcast from P_p.

It turns out that each processor needs to know only parts of the auxiliary vectors $\tilde{w}$ and $\tilde{z}$. In particular, the "local parts" ${\tilde{z}^{(\pm\pi)}}^T$ of the matrix product $\tilde{z}^T = \tilde{v}^T \cdot A_{\beta+1,\beta}$

can be computed without any communication. The same is true for the ensuing rank-2 updates of the subblocks held in each processor.

Finally, to prepare for the next transformation, the block pair—together with its decomposition into subblocks—is shifted by one row and one column. This requires the processors P_π, $\pi = 1, \ldots, p-1$, to send one column to their successors and the processors P_π, $\pi = 2, \ldots, p$, to send one column to their predecessors.

ALGORITHM 4.2. *"$I - vv^T = Q_\beta \rightarrow$ transform $B_\beta \rightarrow Q_{\beta+1} = I - \tilde{v}\tilde{v}^T$," parallel algorithm (one program for all processors P_π, $\pi = 1, \ldots, p$, of the cluster).*

```
                              (* compute $x$ and $\tilde{x}$ *)
$\begin{pmatrix} x \\ \tilde{x} \end{pmatrix} := \begin{pmatrix} S_{1,-\pi} \\ S_{2,-\pi} \end{pmatrix} v^{(-\pi)} + \begin{pmatrix} S_{1\pi} \\ S_{2\pi} \end{pmatrix} v^{(\pi)}$          (* stored elements *)
$x^{(\pm\pi)} := x^{(\pm\pi)} + S'_{1,\pm\pi}\, v$                                (* by transposition *)
fan-in ($\begin{pmatrix} x \\ \tilde{x} \end{pmatrix}$) at $P_p$
                              (* new Householder vector *)
if me = $P_p$
    $h := A_{\beta+1,\beta}(1:d,1) - v_1 \cdot \tilde{x}$                  (* first column of $A_{\beta+1,\beta}Q_\beta$ *)
    determine a Householder vector $\tilde{v}$, $||\tilde{v}|| = \sqrt{2}$, such that
    $(I - \tilde{v}\tilde{v}^T) \cdot h = (*, 0, \ldots, 0)^T$
broadcast $(x, \tilde{x}, \tilde{v})$ from $P_p$ to $P_\pi$ $(1 \le \pi \le p-1)$
                              (* locally needed parts of the auxiliary vectors *)
$\tilde{z}^{(\pm\pi)^T} := \tilde{v}^T S_{2,\pm\pi}$
$w := x - \frac{1}{2}(v^T x) \cdot v$
$\tilde{w}^{(\pm\pi)} := \tilde{z}^{(\pm\pi)} - (\tilde{v}^T \tilde{x}) \cdot v^{(\pm\pi)}$
                              (* rank-2 modifications *)
$S_{1,\pm\pi} := S_{1,\pm\pi} - vw^{(\pm\pi)^T} - wv^{(\pm\pi)^T}$
$S_{2,\pm\pi} := S_{2,\pm\pi} - \tilde{x}v^{(\pm\pi)^T} - \tilde{v}\tilde{w}^{(\pm\pi)^T}$
                              (* shift the subblock partitioning *)
shift the block columns $-\pi$ and $\pi$ by one column/row
if $\pi \ne p$
    send (first column of block column $-\pi$) to $P_{\pi+1}$
if $\pi \ne 1$
    receive (last column of block column $-\pi$) from $P_{\pi-1}$
if $\pi \ne 1$
    send (first column of block column $\pi$) to $P_{\pi-1}$
if $\pi \ne p$
    receive (last column of block column $\pi$) from $P_{\pi+1}$
```

An efficient implementation will not simply substitute this algorithm for the corresponding lines in Algorithm 3.2. To improve performance, it is better to break up the β loops of Algorithm 3.2 and to move them into Algorithm 4.2. More precisely, each processor first computes its contribution to the matrix vector products x and $\tilde{x}$ for *all* of its b "active" block pairs; then these vectors are summed up in a *single* **fan-in** operation. Analogously, the vectors x, $\tilde{x}$, and $\tilde{v}$ for all the active block pairs (and later the columns) are sent together. By packing the communication in this way, we significantly reduce the startup cost but not the amount of data to send.

The "reflected distribution" of the block columns to the processors balances the work load within the cluster fairly well. A detailed analysis [12, Lemma 5.6] reveals that, of the approximately $12d^2$ flops that are necessary to transform a block pair, each processor has a share of $12/p \cdot d^2 \cdot (1+O(d/p))$. Thus, we might hope for nearly full speedup for the transformation of the block pair on a cluster of size p, as compared to a single-processor cluster.

In practice, three factors reduce the attainable speedup. First, the relative cost of communication is higher than for the parallelism between different blocks: each processor communicates $O(d)$ elements for every $O(d^2/p)$ flops, whereas neglecting the parallelism within the block pairs allows $O(ld^2)$ flops for every $O(d)$ elements to communicate. Also, the communication pattern is slightly more complicated. Second, even though the constant in the above O-term is rather small (about 2), the number d/p of columns per processor must be quite large to make the load imbalance negligible. Third, and often most important, the level-2 BLAS may be *very* sensitive to the size and shape of the (sub)matrices with which they are called. If we exploit the parallelism within the block pair, then the bulk of computation can still be handled by calls to the BLAS 2. But now each processor needs to call them more often (16 times in our implementation), and these calls operate on (sub)matrices that are much smaller than the $d \times d$ blocks used in Algorithm 4.1. Thus, the BLAS performance may—but need not—be significantly lower.

Note that the data distribution used in Algorithm 4.1 differs significantly from the cyclic distributions that were proposed for the Householder reduction of *dense* symmetric matrices (e.g., [2]–[4]). In the dense case each transformation affects one row and one column less than the previous transformation. Therefore, optimizing the *overall* load balance requires consideration of not only the layout of the original matrix but also of all the intermediate transformed matrices. Cyclic distribution of rows or columns seems to be appropriate for small numbers of processors, while torus-wrapped distribution is better for large numbers of processors. If the use of level-3 BLAS is essential, then a cyclic distribution of blocks or block columns (block rows) should be used. In contrast to the dense case, our (sub)algorithm repeatedly transforms a block pair of constant size. Thus, we can optimize the *overall* load balance by considering only one single transformation. Apart from the different data layout our (sub)algorithm is similar to those cited above.

5. Numerical experiments on the iPSC/860 hypercube. The numerical experiments were performed on the 8 node and 128 node iPSC/860 hypercube parallel computers at Argonne National Laboratory and at Oak Ridge National Laboratory, respectively.

For all three algorithms the reduction time is nearly independent of the matrix entries. The timings reported in the sequel were made with banded Toeplitz matrices containing δ on the δth subdiagonal, $\delta = 1, \ldots, d$, and 0 on the principal diagonal, and with band matrices containing i in the ith column.

All programs are written in Fortran77, the calls to the BLAS were directed to the optimized assembly-coded BLAS of Kuck and Associates. All computations were done in double precision and do *not* include accumulation of the transformation matrix.

The EISPACK routine BANDR is available via Netlib. As the LAPACK library was not yet public at the time of the experiments, we had to use the final test version of the library; thus, the DSBTRD (double precision version of _SBTRD) results are only preliminary.

The flop counts are exact for the SBTH algorithm with Householder transformations. We use the *same* flop count for the algorithms BANDR and DSBTRD although they actually have different operation counts: BANDR needs about $4n^2d$ flops, DSBTRD and SBTH require roughly $6n^2d$ flops. Thus, the Mflops performance of the EISPACK routine is overestimated by about 50%.

By adopting this "biased" flop count we are able to compare the *execution time* of the library algorithms BANDR and DSBTRD with the new algorithm SBTH (run on one processor), as well as with each other. For example, if the indicated Mflops rate of BANDR is twice that of SBTH, then SBTH needs twice as long to complete. This information seems to be more relevant for practical use than mere Mflops throughput.

From the Mflops/processor figures for parallel execution of the algorithm SBTH, it is easy to obtain two other figures often reported in the literature: the overall performance (just multiply by the number of processors in use) and the speedup (divide the overall parallel performance by the performance on one processor).

Figures 4–8 summarize measurements for the parallel algorithm when only the parallelism between different blocks is exploited. As pointed out in §4, this allows for a simpler and more efficient calling sequence to the BLAS.

Figure 4 comprises the results for matrices of (semi)bandwidth $d = 5$ and varying matrix dimension n. The curves indicate that the EISPACK routine is about two times faster than the LAPACK counterpart and three times faster than our algorithm, run on a single processor. But, if the latter is run on multiple processors, and if each processor holds a "sufficient" number of columns (about 100), we obtain nearly full speedup. Thus, the parallel algorithm will be superior if it is run on four or more processors.

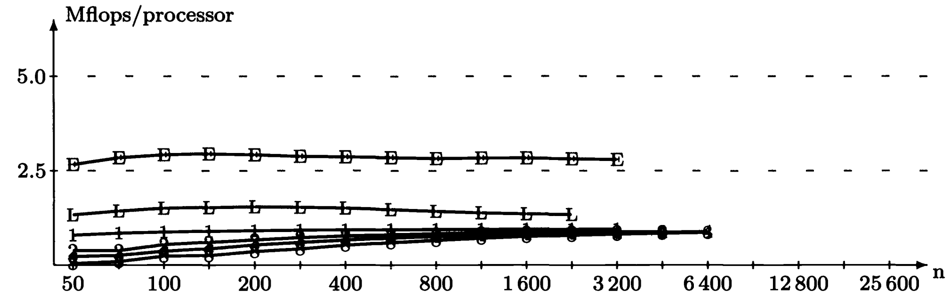


FIG. 4. *Performance of the reduction algorithms for (semi)bandwidth d = 5. The "E" and "L" curves correspond to the algorithms of the EISPACK and LAPACK library, respectively, which are run on one processor. The curves "1", "2", "4," and "8" (from top to bottom) correspond to the parallel algorithm run on the respective number of processors (parallelism between different block pairs only).*

For $d = 10$ the situation has slightly changed. Now, the parallel algorithm has caught up with the LAPACK routine even if it is run on a single processor. Both perform at about 2–2.5 Mflops while the EISPACK routine still leads with some 3 Mflops.

It is noteworthy that the *absolute time* for reducing a matrix with semibandwidth 10 with the new algorithm is *less* than for a matrix of the same order with semibandwidth 5, although nearly twice as many operations are needed.

This is due to the varying performance of the underlying BLAS. For very small blocks ($d = 5$), the overhead for calling a subroutine more than outweighs the gain from assembly coding within the BLAS, and, therefore, a Fortran program with all arithmetic coded in loops would be about twice as fast as the BLAS-based code we used for our timings. With increasing semibandwidth, however, there are *fewer* calls to the BLAS, and the blocks are bigger. Thus, for $d = 10$ our program is competitive with its loop-based counterpart.

For $d = 20$—and all values of n—the situation has turned completely in favor of the new algorithm (see Fig. 5). It now achieves about 5 Mflops on one processor while the EISPACK and LAPACK routines are well below.

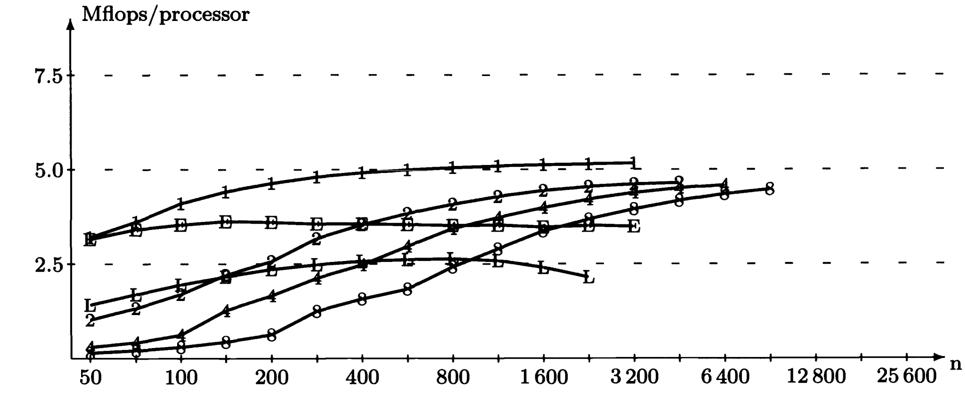


FIG. 5. *Performance of the reduction algorithms for (semi)bandwidth $d = 20$. For* SBTH, *only parallelism between different block pairs is exploited.*

Again, the absolute time for reducing a matrix of semibandwidth 20 with SBTH is less than for a matrix of the same order and $d = 10$. This still is an effect of increasing BLAS performance: doubling the blocksize from 5 to 10 more than doubles the Mflops performance of the BLAS, and it does again for $d = 10/d = 20$.

With increasing semibandwidth, the performance of the new algorithm further increases (Figs. 6 and 7) and reaches nearly 15 Mflops on one processor for $d = 113$, which is about four times the highest performance we were able to obtain from the EISPACK and LAPACK routines.

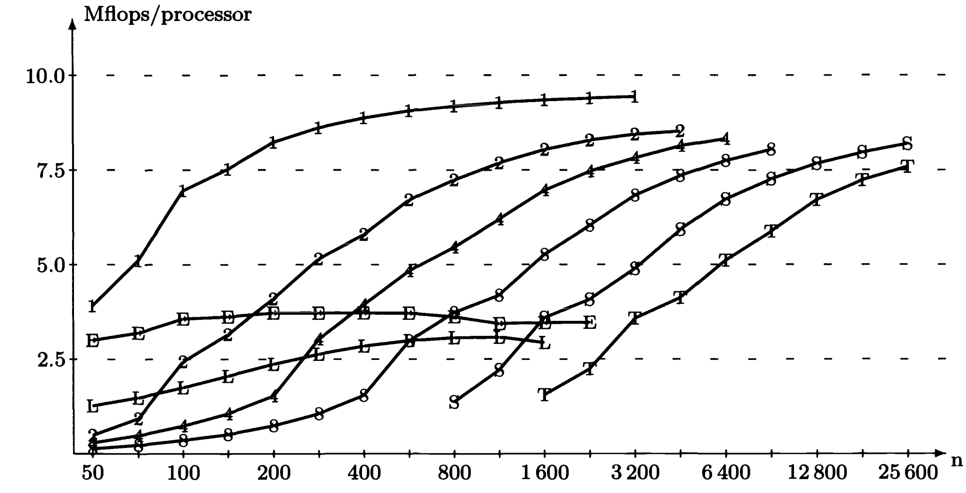


FIG. 6. *Performance of the reduction algorithms for (semi)bandwidth $d = 40$. The curves "S" and "T" correspond to the parallel algorithm run on 16 and 32 processors, respectively. For* SBTH, *only parallelism between different block pairs is exploited.*

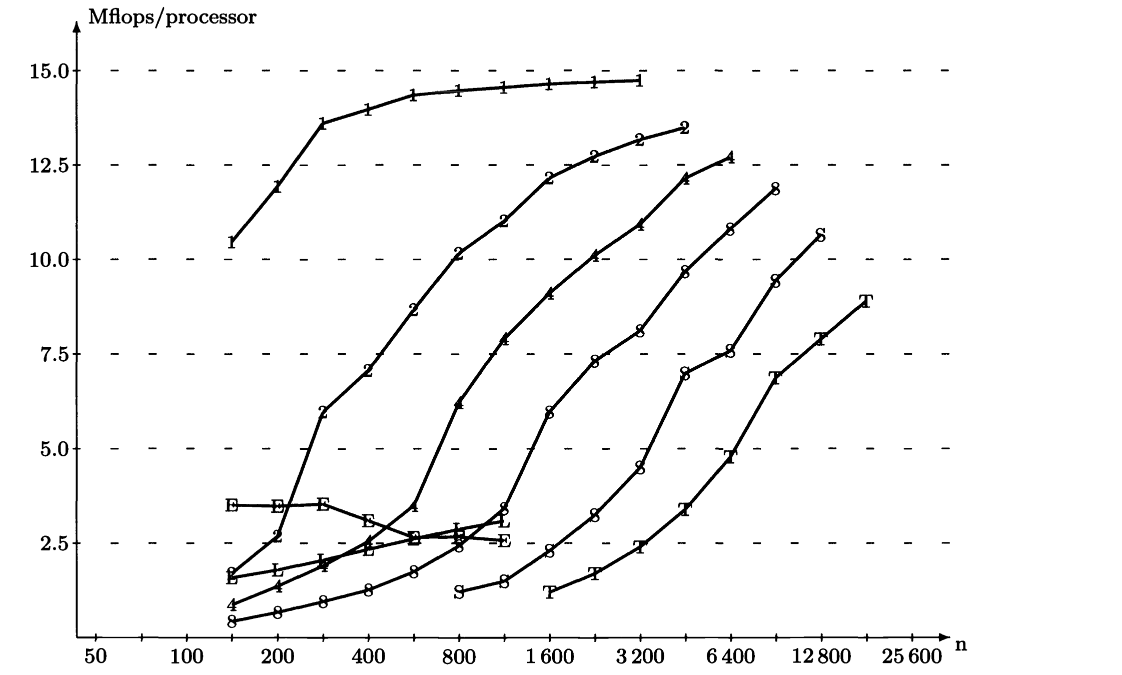


FIG. 7. *Performance of the reduction algorithms for (semi)bandwidth* $d = 113$. *For* SBTH, *only parallelism between different block pairs is exploited.*

In order to provide at least some representative "raw" Mflops and speedup figures we summarize some of the data of Fig. 6 in Table 1. For matrix sizes $n \geq 3\,200$ the serial time had to be estimated; this is indicated by a "$\sim$".

TABLE 1

Performance of the routines BANDR from EISPACK and DSBTRD from LAPACK and comparison with the parallel algorithm SBTD, *run on* 1, 8, *and* 32 *processors. For* SBTH, *only parallelism between different block pairs is exploited. The (semi)bandwidth is* $d = 40$.

		$n =$							
		200	400	800	1 600	3 200	6 400	12 800	25 600
EISPACK	Mflops	3.7	3.7	3.6	3.5	$\sim$ 3.5			
LAPACK	Mflops	2.4	2.8	3.1	2.9	$\sim$ 2.9			
SBTD 1 proc	Mflops	8.2	8.9	9.2	9.4	9.4	$\sim$ 9.5		
	speedup vs EISPACK	2.2	2.4	2.5	2.7	$\sim$ 2.7			
	speedup vs LAPACK	3.5	3.1	3.0	3.2	$\sim$ 3.3			
SBTD 8 proc	Mflops	6.0	12.3	29.9	42.1	54.6	62.0		
	speedup vs EISPACK	1.6	3.3	8.2	12.2	$\sim$ 16	$\sim$ 18		
	speedup vs LAPACK	2.5	4.3	9.7	14.3	$\sim$ 19	$\sim$ 21		
	speedup vs SBTD(1)	0.7	1.4	3.2	4.5	5.8	$\sim$ 6.5		
SBTD 32 proc	Mflops				49.9	115	164	215	243
	speedup vs EISPACK				14.4	$\sim$ 33	$\sim$ 47	$\sim$ 61	$\sim$ 69
	speedup vs LAPACK				17.2	$\sim$ 39	$\sim$ 54	$\sim$ 74	$\sim$ 84
	speedup vs SBTD(1)				5.3	12.1	$\sim$ 17	$\sim$ 23	$\sim$ 26

To demonstrate that the parallel algorithm *scales* well if the problem size is increased with the number of processors, we draw the data for $d = 80$ in a slightly different way: now we plot the performance against the wrap ω (i.e., the number of logical blocks *per processor*) instead of the matrix dimension n. Figure 8 then indicates that the performance per processor is almost independent of the number of processors as long as we keep the wrap constant. In fact, the curves for 4, 8, 16, and 32 processors nearly coincide. The curves also confirm very well the speedup prediction which was given at the end of §3. If the band is not long enough to reach all the processors ($\omega < 1$), then the speedup is proportional to the wrap (which, in this case, is just the percentage of processors holding a portion of the band). As soon as all the processors are contributing to the reduction process ($\omega \geq 1$), the speedup is approximately $p/(1 + 1/\omega)$.

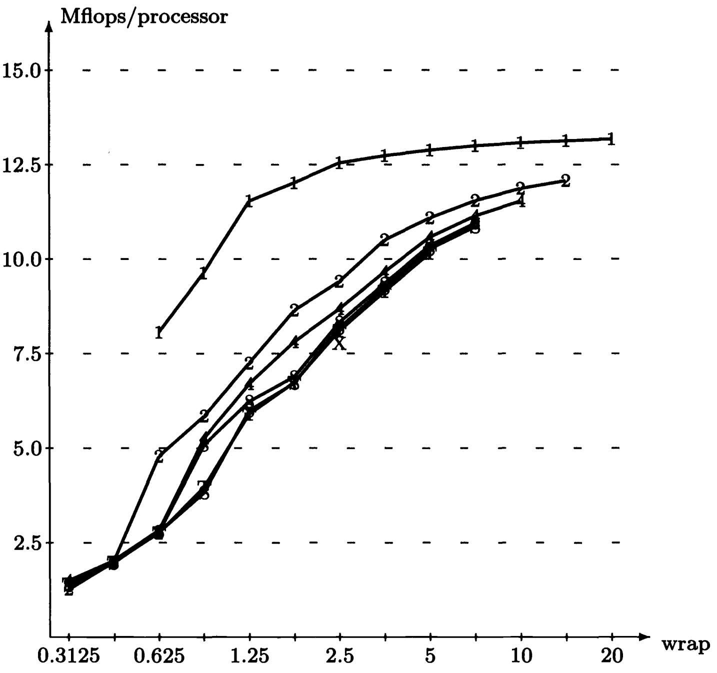


FIG. 8. *Performance of the algorithm* SBTH *for (semi)bandwidth $d = 80$ as a function of the wrap (number of logical blocks per processor). The "X" indicates a single run on* 64 *processors. Only parallelism between different block pairs is exploited.*

Finally, we investigate the performance of the algorithm if the parallelism within the blocks is also exploited. For the runs collected in Fig. 9 we keep the number of processors constant at eight and vary their grouping into clusters, from eight single-processor clusters to a single cluster consisting of eight processors. As expected, the "one processor per cluster" version performs best if the number of logical blocks in the band is large enough. As mentioned earlier, the level-2 BLAS on the iPSC/860 put a severe penalty on using small (sub)matrices in their calls. This explains the massive drop in performance for larger matrices if we increase the number of processors in each cluster from $p = 1$ to $p = 2$, the Mflops per processor are nearly halved! Remember that in Algorithm 4.1 the BLAS are called with (square) blocks of size d, whereas the width of the subblocks in Algorithm 4.2 is only $d/(2p)$ and their height ranges from $d/(2p)$ to d. Even so, if the number of logical blocks is smaller than the number of processors, then it may pay to reduce the number of clusters (and thereby to increase the wrap) by using clusters with more than one processor. In fact, for $n \approx 500, \ldots, 800$ the configuration into four clusters with two processors each is optimal. This superiority of the case $p > 1$ would be more pronounced if the BLAS performance were better balanced with respect to the matrix size.

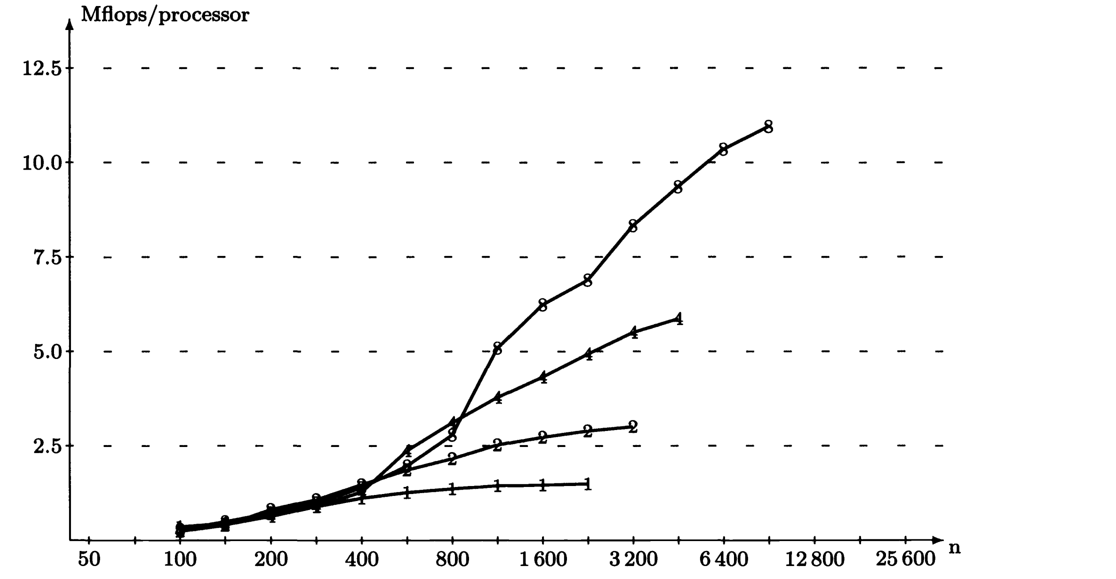


FIG. 9. *Performance of the algorithm* STBH *if also the parallelism within the blocks is exploited. The overall number of processors is* eight *for all the curves. The digits indicate the number of clusters in use. For example, the "4" curve corresponds to four clusters of two processors each.*

6. Concluding remarks. A detailed error analysis [12, Thm. 4.3] shows that if the computations are carried out in finite precision arithmetic with a relative error $\leq \vartheta$ in each operation, then the computed tridiagonal matrix $\bar{T}$ is *exactly* orthogonally similar to a symmetric (but generally not banded) matrix $\bar{A}$ with

$$||\bar{A} - A||_F = O(nd) \cdot \vartheta \cdot ||A||_F.$$

Here, $|| \cdot ||_F$ denotes the Frobenius norm of a matrix. By the Wielandt–Hoffman theorem [16, p. 104], this implies that the eigenvalues $\bar{\lambda}_i$ of the computed tridiagonal matrix cannot differ too much from the eigenvalues λ_i of the original band matrix A:

$$\left(\sum_{i=1}^{n}(\bar{\lambda}_i - \lambda_i)^2\right)^{1/2} \leq O(nd) \cdot \vartheta \cdot ||A||_F.$$

Therefore, the parallel reduction algorithm is numerically stable. The error bounds for the EISPACK and LAPACK algorithms are of the same form, although with smaller constants in the O-term [12, Thm. 4.2].

In this paper we do not deal with updating the transformation matrix, an option available in the EISPACK and LAPACK routines. It is, however, not very difficult to include these updates in our parallel algorithm.

In [12] we also developed a parallel algorithm based on (fast) Givens rotations, which is somewhat nearer to the EISPACK and LAPACK routines than the method discussed in this paper. This algorithm also relies on the block decompositions introduced in §2 and on the "parallel frame" given in Algorithm 3.2. It combines the advantages of the EISPACK routine (lower memory requirements and lower operation count) with a medium-grained parallelism, but the arithmetic must be done with the level-1 BLAS. In addition, although there is still some potential for parallelism within the blocks, it is much more difficult to utilize than with the Householder approach.

To conclude, the algorithm SBTH seems very promising on the iPSC/860 hypercube, as it should be on all machines with optimized BLAS-2 kernels and low bandwidth from the main memory to the arithmetical unit(s). The algorithm performs well if it is run on a single processor, and also it offers nearly full speedup if the band is long enough to be wrapped several times onto the processors.

Acknowledgments. This paper summarizes and extends major parts of my Ph.D. thesis. I am very grateful to my advisors G. Alefeld and A. Frommer for their support and advice. Many thanks to Argonne National Laboratory and to Oak Ridge National Laboratory for providing access to their hypercube parallel computers. Special thanks to Christian Bischof for initiating my visit at Argonne National Laboratory. I would also like to thank the unknown referees for their valuable comments.

REFERENCES

[1] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK User's guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[2] H. Y. CHANG, S. UTKU, M. SALAMA, AND D. RAPP, *A parallel Householder tridiagonalization stratagem using scattered row decomposition*, Internat. J. Numer. Methods Engrg., 26 (1988), pp. 857–873.

[3] ———, *A parallel Householder tridiagonalization stratagem using scattered square decomposition*, Parallel Comput., 6 (1988), pp. 297–311.

[4] J. DONGARRA AND R. V. VAN DE GEIJN, *LAPACK Working Note 30—Reduction to Condensed Form for the Eigenvalue Problem on Distributed Memory Architectures*, Tech. Rep., Computer Science Department, Univ. of Tennessee, Knoxville, 1991.

[5] J. J. DONGARRA, J. DU CROZ, S. HAMMARLING, AND I. DUFF, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[6] 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.

[7] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139–s154.

[8] B. S. GARBOW, J. M. BOYLE, J. J. DONGARRA, AND C. B. MOLER, *Matrix Eigensystem Routines—EISPACK Guide Extension*, Springer-Verlag, Berlin, 1977.

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

[10] L. KAUFMAN, *Banded eigenvalue solvers on vector machines*, ACM Trans. Math. Software, 10 (1984), pp. 73–86.

[11] B. LANG, *Matrix vector multiplication for symmetric matrices on parallel computers and applications*, Z. Angew. Math. Mech., 71 (1991), pp. T807–T808.

[12] ———, *Parallele Reduktion symmetrischer Bandmatrizen auf Tridiagonalgestalt*, Ph.D. thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, 1991.

[13] C. L. LAWSON, R. J. HANSON, R. J. KINCAID, AND F. T. KROGH, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Software, 5 (1979), pp. 308–323.

[14] H. RUTISHAUSER, *On Jacobi rotation patterns*, in Proc. Symposia Appl. Math., Vol. 15, Experimental Arithmetic, High Speed Computing and Mathematics, Providence, RI, 1963, pp. 219–239.

[15] H. R. SCHWARZ, *Tridiagonalization of a symmetric band matrix*, Numer. Math., 12 (1968), pp. 231–241.

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

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1339–1358, November 1993

COMPUTING PERIODIC GRAVITY WAVES ON WATER BY USING MOVING COMPOSITE OVERLAPPING GRIDS*

N. ANDERS PETERSSON†

Abstract. The composite overlapping grid method is applied to compute periodic gravity waves on water of finite constant depth. One component grid is made to follow the free surface while the remaining components are independent of the location of the surface. A pseudo-arclength continuation method is used to compute the solution as function of the phase velocity of the wave. The type of equation associated with some grid points and the number of equations in the discretized problem will change when the surface moves. The author expounds a stable way of switching the composite grid during the continuation procedure that works close to limit points. An adaptive technique is also developed to efficiently resolve the solution where sharp gradients develop.

Numerical examples are presented that show very good agreement with existing results.

Key words. adaptive grid, composite overlapping grid, pseudo-arclength continuation, water wave

AMS subject classifications. 65N50, 76B15

1. Introduction.

We consider two-dimensional irrotational gravity waves moving with constant phase velocity on water of finite constant depth. Both the infinitely deep case [2], [10] and the present case [4], [14] have been previously studied extensively. The aim of the research described here is to develop an accurate method that can be easily extended to compute the subcritical flow around an underwater obstacle, where we feel that the existing methods [9] still need improvement. We remark that an efficient method, based on a boundary integral formulation, has been devised for the special case when the flow behind the obstacle is supercritical, i.e., when waves are absent behind the obstacle [5]. This paper can be seen as the next step from [11] towards the solution of the subcritical problem. We prefer to first develop the method for the periodic case to be able to make comparisons with existing results.

We apply the composite overlapping grid method [3]. The basic idea is to make one component grid follow the free surface and let the remaining components be independent of the location of the surface. To resolve sharp gradients in the solution we also develop an adaptive technique. The adaptation is done locally on each component grid by changing the number of grid points and the stretching function. We would like to point out that the adaptive, moving grid approach developed here is general to problems where the shape of the domain is a function of the solution and/or the solution develops sharp gradients. One advantage of the composite overlapping grid method compared to the more commonly used boundary integral techniques is that the composite overlapping grid approach can also be used if effects of vorticity and viscosity are included. Those effects are not straightforward to include in a boundary integral formulation.

We use a pseudo-arclength continuation technique [8] to compute the solution as function of the phase velocity. When the surface moves, the type of equation associated with some grid points and the number of equations in the discretized problem change. The basic method therefore needs to be modified to allow for changes in the composite grid during the continuation procedure. A stable way of doing this that works close to limit points is devised.

*Received by the editors March 23, 1992; accepted for publication (in revised form) November 11, 1992. This work was supported by Office of Naval Research grants N-00014-90-J-1695 and N-00014-90-J-1382 and by the U.S. Department of Energy through Los Alamos National Laboratory.

†Department of Mathematics, University of California, Los Angeles, California 90024. Present address: Center for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

The remainder of the paper is organized as follows. In §2 we scale the problem and present the governing equations. We discuss how to construct the composite overlapping grid in §3 and the problem is discretized in §4. Thereafter, in §5, we present the continuation method. A stable way of switching composite grid is presented in §6. Here, we also describe how the resolution of the grid is adapted to the solution. In §7 we discuss some implementational issues and we make numerical comparisons with existing results in §8. Very good agreement is found.

2. The governing equation. We describe the motion in Cartesian coordinates in a frame of reference fixed with respect to the wave, where the x-axis points opposite to the forward velocity and the z-axis is directed vertically upwards. We assume the motion to be steady in this coordinate system. Let the phase velocity be U and the wavelength be λ, which is defined as the shortest period of the wave. We scale the physical quantities by the length $\lambda/2\pi$ and the velocity $\sqrt{g\lambda/2\pi}$, where g is the acceleration of gravity. In the scaled variables, $z = -d$ corresponds to the bottom and $z = 0$ to the undisturbed free surface. We split the total velocity potential into a free stream potential plus a perturbation potential: $\Phi(x,z) = \mu x + \phi(x,z)$, where $\mu = U/\sqrt{g\lambda/2\pi}$ is the scaled phase velocity. The perturbation potential is governed by (cf. [15])

$$(1) \qquad \Delta\phi = 0, \quad -d \le z \le \eta(x), \quad -\infty < x < \infty,$$

where $\eta(x)$ is the elevation of the free surface. The perturbation potential is subject to the boundary condition

$$(2) \qquad \phi_z(x,-d) = 0, \qquad -\infty < x < \infty.$$

Furthermore, at the free surface we require both the Bernoulli equation and the kinematic condition to be satisfied,

$$(3) \qquad \mu\phi_x + \frac{1}{2}(\phi_x^2 + \phi_z^2) + \eta = 0, \quad -\infty < x < \infty, \quad z = \eta(x),$$

$$(4) \qquad (\mu + \phi_x)\eta_x - \phi_z = 0, \quad -\infty < x < \infty, \quad z = \eta(x).$$

The choice of constant in the right-hand side of (3) fixes the origin in z to the level of the free surface where the velocity, $\sqrt{(\mu+\phi_x)^2 + \phi_z^2}$, equals the phase velocity, μ.

The problem is to find the perturbation potential ϕ and the surface elevation η as functions of the phase velocity μ. We are interested in solutions that are l-periodic in the x-direction, with $l = 2\pi k$, $k = 1, 2, \dots$. The surface elevation will be studied in the interval $0 \le x \le l$ and the perturbation potential in the domain $(x,z) \in \Omega$, $\Omega: \ 0 \le x \le l, \ -d \le z \le \eta(x)$. A convenient measure of a solution is its wave height, which is defined as the vertical distance between the highest crest and the deepest trough of the wave.

A solution of (1)–(4) at a fixed μ is not isolated. This is due to the Galilean invariance in the x-direction and the fact that ϕ is only determined up to a constant. Denote a solution (ϕ,η) by ψ and let $\mathcal{L}[\psi,\mu]$ be the operator described by (1)–(4). Expanding $\mathcal{L}$ around a solution $\psi^{(0)}$ yields

$$(5) \qquad \mathcal{L}[\psi^{(0)} + \psi', \mu] = \mathcal{L}[\psi^{(0)}, \mu] + \mathcal{L}_\psi[\psi^{(0)}, \mu]\psi' + \mathcal{O}(|\psi'|^2).$$

The two degrees of freedom correspond to two zero eigenvalues of $\mathcal{L}_\psi$. It is easy to see that the eigenfunction connected to the undetermined constant is $\phi' = \text{const.}$, $\eta' = 0$. The eigenfunction corresponding to the Galilean invariance is the x-derivative of the present solution, i.e., $\phi' = \phi_x^{(0)}$, $\eta' = \eta_x^{(0)}$.

3. The composite overlapping grid. The composite overlapping grid method is a general tool for solving partial differential equations (PDEs) on complex domains (cf. [3]). The basic idea is to divide the complex domain into simple overlapping subdomains, the union of which completely covers the region of interest. Each subdomain is covered by a structured component grid and the set of component grids taken together is called the composite grid.

There are three different kinds of grid points in the composite grid: discretization, interpolation, and unused grid points. The discrete version of the differential equation or the boundary condition operator is applied at a discretization point. The interpolation points are situated at an interior boundary where the subdomains overlap. Here, the solution is interpolated from an overlapping component grid. The unused points are, as the name indicates, not used in the discretization of the differential equation. Henceforth, the discretization plus the interpolation points will be called the grid points used in the composite grid.

From previous investigations of the present problem, it is known that the spatial scales of the solution can be very small close to the surface. The scales are also known to grow rapidly with the distance from the surface. To resolve the solution without wasting grid points, we will use a coarse component grid close to the bottom, a fine component grid close to the surface, and component grids of intermediate grid sizes in between.

The component grid that includes the surface will be called the surface grid. To make the discretization of the boundary conditions (3), (4) straightforward and accurate, we locate the uppermost grid line of the surface grid at the position of the surface, $\eta(x)$. In order to keep the dependence on η close to the surface and simplify the discretization, we choose the remaining component grids to be Cartesian and independent of η.

Let there be Q component grids and denote by Ω_q the subdomain in the (x, z)-plane that corresponds to grid number q. The component grids are numbered beginning at the bottom and increasing towards the surface (see Fig. 1). The grid number equals the priority of that component in the composite grid. The composite grid will primarily use grid points from the component with the highest priority, secondarily from the grid with the second highest priority, and so on.

3.1. The mapping functions. For each subdomain, we will design a one-to-one mapping function that transforms the unit square in the (r, s)-plane onto Ω_q. These mapping functions are used to construct the component grids by transforming the grid points in a Cartesian grid in the (r, s)-plane onto the (x, z)-plane. In the following, the (r, s)-plane will also be called the parameter space.

For the Cartesian component grids, the mapping functions are simply

$$x^q(r) = lr, \tag{6}$$

$$z^q(r, s) = a_q(1 - s) + b_q s, \tag{7}$$

where a_q and b_q are constants; q is the grid number, $1 \leq q \leq Q - 1$.

To make the dependence of η in the transformed Laplace equation reasonably simple, we choose the mapping function for the surface grid to be

$$x^Q(r) = lt_Q(r), \tag{8}$$

$$z^Q(r, s) = \eta(lt_Q(r)) - b_Q(1 - u_Q(s)). \tag{9}$$

Here, b_Q is the constant vertical thickness of the grid. The function $t_Q(r)$ clusters grid points in a layer around the crest of the wave and the function $u_Q(s)$ is used to concentrate grid points in a layer close to the surface. For example, $t_Q(r) = R^{-1}(r)$, where $R(t) = (t + U_p(t) - U_p(0))/(1 + U_p(1) - U_p(0))$ and

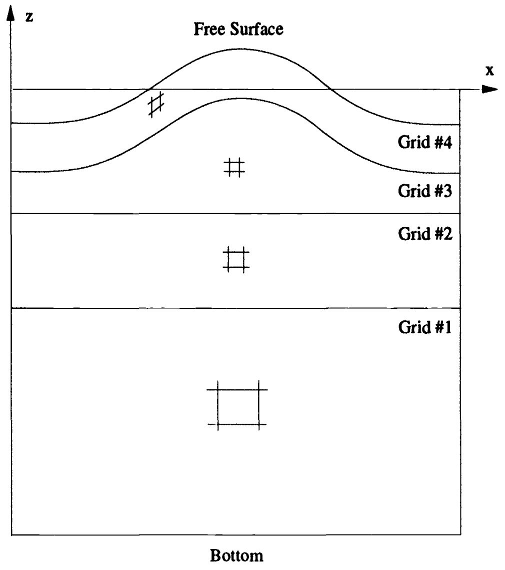


FIG. 1. *The numbering of the component grids. In this case, $Q = 4$.*

$$U_p(t) = \sum_{\nu=-\infty}^{\infty} U(t+\nu), \; U(t) = \frac{\alpha_r}{2} \tanh \beta_r (t - \gamma_r). \tag{10}$$

The stretching is concentrated around $t = \gamma_r$ and the parameter β_r determines the ratio between the smallest and the largest grid size. The constant α_r governs the width of the layer, i.e., the number of grid points that will be in the layer compared to the number outside it. The derivatives of the function $u_Q(s)$ are not required to be periodic. Therefore, the function U replaces U_p in the corresponding expression.

The grid points of grid q are given by $x_j^q = x^q(r_j^q)$ and $z_{j,k}^q = z^q(r_j^q, s_k^q)$, where

$$r_j^q = (j-1)h_r^q, j = 1, 2, \dots, N_q, \tag{11}$$
$$s_k^q = (k-1)h_s^q, k = 1, 2, \dots, M_q, \tag{12}$$

with $h_r^q = 1/(N_q - 1)$ and $h_s^q = 1/(M_q - 1)$; $N_q > 1$ and $M_q > 1$ are natural numbers.

4. Discretizing the equations. In this section, we will describe how the Laplace equation and the boundary conditions are discretized by a second-order accurate scheme on a composite grid. We adopt the mapping method which uses the previously constructed mapping functions to form the discrete set of equations on each component grid. The component grid problems are coupled to each other by interpolation relations at the interior boundaries where the subdomains overlap.

We begin with some definitions. We define a component grid function on grid q by $g_{j,k}^q = g(x_j^q, z_{j,k}^q)$ and a surface grid function on grid Q by $f_j = f(x_j^Q)$. The forward, backward, and central divided difference operators in the r-direction on grid q are defined as

$$D_{+j} g_{j,k}^q = \frac{g_{j+1,k}^q - g_{j,k}^q}{h_r^q},$$
$$D_{-j} g_{j,k}^q = D_{+j} g_{j-1,k}^q,$$

$$D_{0j}g^q_{j,k} = \frac{1}{2}\left(D_{+j} + D_{-j}\right)g^q_{j,k}.$$

A corresponding notation is used in the s-direction.

On the Cartesian grids, we discretize (1) by

$$\left(l^{-2}D_{+j}D_{-j} + (b_q - a_q)^{-2}D_{+k}D_{-k}\right)\phi^q_{j,k} = 0. \tag{13}$$

This stencil is applied to the discretization points with indices in the range $1 \leq j \leq N_q - 1$, $2 \leq k \leq M_q - 1$. The periodicity in the x-direction implies periodicity in the r-direction for all component grid functions. It is enforced by $\phi^q_{0,k} = \phi^q_{N_q-1,k}$ and $\phi^q_{N_q,k} = \phi^q_{1,k}$. We discretize the boundary condition at the bottom (2) by the second-order accurate one-sided formula

$$(b_1 - a_1)^{-1}\left(D_{+k} - \frac{h^1_s}{2}D_{+k}D_{+k}\right)\phi^1_{j,k} = 0, \tag{14}$$

for $k = 1$ and $1 \leq j \leq N_1 - 1$.

To discretize the Laplace equation on the surface grid, we transform it to the parameter space and replace the derivatives of ϕ in the resulting expression by second-order accurate divided differences. This yields

$$\left(\mathcal{A}_{j,k}D_{+j}D_{-j} + \mathcal{B}_{j,k}D_{0j}D_{0k} + \mathcal{C}_{j,k}D_{+k}D_{-k} + \mathcal{D}_{j,k}D_{0j} + \mathcal{E}_{j,k}D_{0k}\right)\phi^Q_{j,k} = 0, \tag{15}$$

for the discretization points in $1 \leq j \leq N_Q - 1$, $2 \leq k \leq M_Q - 1$. The coefficients are (cf. [13])

$$\mathcal{A} = (r^Q_x)^2 + (r^Q_z)^2, \quad \mathcal{B} = 2(r^Q_x s^Q_x + r^Q_z s^Q_z), \quad \mathcal{C} = (s^Q_x)^2 + (s^Q_z)^2, \tag{16}$$

and

$$\mathcal{D} = r^Q_{xx} + r^Q_{zz}, \; \mathcal{E} = s^Q_{xx} + s^Q_{zz}. \tag{17}$$

Here, r^Q and s^Q are the inverses of the mapping functions x^Q and z^Q, respectively, evaluated at the grid point in question. The subscripts on the metric quantities denote partial differentiation, i.e., $r^Q_x = \partial r^Q/\partial x$, etc.

The boundary conditions at the free surface are discretized by the same technique as the Laplace equation. This yields

$$\begin{aligned}\mu(r^Q_x D_{0j} + s^Q_x D_{1k})\phi^Q_{j,k} &+ \frac{1}{2}\left((r^Q_x D_{0j} + s^Q_x D_{1k})\phi^Q_{j,k}\right)^2 \\ &+ \frac{1}{2}\left((r^Q_z D_{0j} + s^Q_z D_{1k})\phi^Q_{j,k}\right)^2 + \eta_j = 0,\end{aligned} \tag{18}$$

$$\left(\mu + (r^Q_x D_{0j} + s^Q_x D_{1k})\phi^Q_{j,k}\right) r^Q_x D_{0j}\eta_j - \left(r^Q_z D_{0j} + s^Q_z D_{1k}\right)\phi^Q_{j,k} = 0. \tag{19}$$

In these formulae we used $D_{1k} = D_{-k} + h_s D_{-k}D_{-k}/2$, which is a second-order accurate approximation of the s-derivative. We apply (18), (19) to $k = M_Q$ and $1 \leq j \leq N_Q - 1$.

The component grid functions are coupled by interpolation relations. Our difference scheme is a second-order accurate approximation of a second-order equation and the component grids have an overlap which is proportional to the grid size. To get second-order accuracy for the total solution, it is necessary to use (at least) third-order accurate interpolation (cf. [3]). For example, let $\phi^a_{n,m}$ be an interpolation point with interpolation location (j, k, b). This means that $\phi^a_{n,m}$ will be interpolated from $\phi^b_{j+p,k+q}$,

$-1 \le p, q \le 1$. Let $(\tilde{r}, \tilde{s})$ satisfy $x^a_n = x^b(\tilde{r})$ and $z^a_{n,m} = z^b(\tilde{r}, \tilde{s})$. The biquadratic interpolation yields

$$\phi^a_{n,m} - \sum_{p=-1}^{1} \sum_{q=-1}^{1} \alpha_p(\tilde{r})\beta_q(\tilde{s})\phi^b_{j+p,k+q} = 0, \tag{20}$$

where α_p and β_q are quadratic polynomial base-functions.

5. Continuation in μ. We will use a pseudo-arclength continuation method to calculate (ϕ, η) as function of the phase velocity μ. The basic method described in [8] cannot be used directly due to the two degrees of freedom that make the linearized operator singular. The method also needs to be modified because the number of grid points used in the composite grid depends on the surface elevation. However, the discussion of that aspect will be postponed to §6.

To handle the two degrees of freedom, one can add two new degrees of freedom and two extra equations such that the Jacobian of the extended problem becomes nonsingular [7]. However, it is possible to avoid adding extra unknowns and equations by instead enforcing the increments in the Newton iteration to be orthogonal to the nullspace connected to the two zero eigenvalues. This is equivalent to solving (1)–(4) in the subspace orthogonal to the nullspace. The part of the solution in the nullspace is of no interest, since it only corresponds to adding a constant to the perturbation potential and shifting the solution in the x-direction.

To enforce the orthogonality of the increments in the Newton iteration, we also need the nullspace of the adjoint operator. Unfortunately, it is difficult to numerically compute accurate approximations of these nullspaces. To overcome this difficulty, we modify the boundary condition at the bottom to fix the constant part of the solution. Instead of (2), we use

$$\phi_z(x, -d) + \phi(0, -d) = 0, \qquad 0 \le x \le l. \tag{21}$$

The discrete boundary condition (14) is modified in a corresponding way. The idea is that a solution with $\phi(0, -d) = 0$ satisfies the original boundary condition without introducing an undetermined constant. The linearized modified operator will therefore only have one zero eigenvalue. This makes an accurate computation of the corresponding nullspace much easier. Due to nonconservation, we cannot expect $\phi(0, -d)$ to be zero in the discrete approximation. Instead, it will get a small value that indicates how well the continuous conservation property is satisfied. We regard this value as a measure of the accuracy of the discrete solution.

The discrete set of equations can be written in abstract form as $L[u, \mu] = 0$, where $L : \mathcal{X} \times \Re \to \mathcal{X}$. Here, $\mathcal{X}$ is an n-dimensional vectorspace and n is the number of grid points used in the composite grid plus the number of grid points along the surface. The vector u contains ϕ at every used grid point and η at the grid points along the surface. We define a scalar product and a norm for $x, y \in \mathcal{X}$ by

$$\langle x, y \rangle = \frac{1}{n} \sum_{i=1}^{n} x_i y_i, \qquad \|x\| = \sqrt{\langle x, x \rangle}. \tag{22}$$

The eigenvalue connected to the Galilean invariance will henceforth be called the shift-eigenvalue. Let e_r and e_l denote the right and left eigenvectors of the Jacobian matrix $\partial L / \partial u$ corresponding to the shift-eigenvalue. Let the eigenvectors be normalized to have $\|e_r\| = 1$ and $\langle e_l, e_r\rangle = 1$. The projection P maps $\mathcal{X}$ onto the eigenspace according to $Pf = \langle e_l, f\rangle e_r$, $f \in \mathcal{X}$.

We will consider solving

$$(I - P[u,\mu])L[u,\mu] = 0, \tag{23}$$

for part of the solution in $(I - P)\mathcal{X}$. The part in $P\mathcal{X}$ corresponds to the phase of the solution and it will be determined during the solution procedure. A point (u, μ) that satisfies $(I - P[u,\mu])L[u,\mu] = 0$ will be called a solution point and we will call a point nonsingular if all eigenvalues of $\partial L/\partial u$ except the shift-eigenvalue are bounded away from zero.

5.1. The nonsingular case. Following [8], the solution (u, μ) will be considered as a function of the pseudo-arclength σ, $u = u(\sigma)$ and $\mu = \mu(\sigma)$. Assume that a nonsingular solution point (u_0, μ_0) is known, and let it have pseudo-arclength σ_0. We define the pseudo-arclength relative to that point by

$$\sigma = \sigma_0 + \langle \dot u_0, u - u_0\rangle + \dot\mu_0(\mu - \mu_0). \tag{24}$$

The tangent $(\dot u_0, \dot\mu_0)$ is the solution of

$$L_u[u_0, \mu_0]\dot u_0 = -L^I_\mu[u_0, \mu_0]\dot\mu_0. \tag{25}$$

Here we used the notation $L^I_\mu[u_0,\mu_0] = (I - P[u_0,\mu_0])L_\mu[u_0,\mu_0]$. Equation (25) is solved under the side-condition $P\dot u_0 = 0$ and the normalization $\|\dot u_0\|^2 + |\dot\mu_0|^2 = 1$. We also require the scalar product between the previous and the present tangent to be positive.

To get the same number of equations as dependent variables, we augment $(I - P[u,\mu])L[u,\mu] = 0$ by the arclength equation $N[u,\mu;\sigma] = 0$, where

$$N[u,\mu;\sigma] = \langle \dot u_0, u - u_0\rangle + \dot\mu_0(\mu - \mu_0) - (\sigma - \sigma_0). \tag{26}$$

We use the predictor $(u^0, \mu^0) = (u_0 + \dot u_0\Delta\sigma, \mu_0 + \dot\mu_0\Delta\sigma)$ as initial guess for the solution at $\sigma = \sigma_0 + \Delta\sigma$. The predictor is corrected by Newton's method on the augmented system, where the improvements of the solution are found by solving

$$\begin{pmatrix} L_u[u^k,\mu^k] & L^I_\mu[u^k,\mu^k] \\ N_u[u^k,\mu^k] & N_\mu[u^k,\mu^k] \end{pmatrix} \begin{pmatrix} \Delta u^k \\ \Delta\mu^k \end{pmatrix} = -\begin{pmatrix} L^I[u^k,\mu^k] \\ N[u^k,\mu^k] \end{pmatrix} \tag{27}$$

by the bordering algorithm, under the side-condition $P[u^k,\mu^k]\Delta u^k = 0$. The solution is then updated according to

$$u^{k+1} = u^k + \Delta u^k, \tag{28}$$

$$\mu^{k+1} = \mu^k + \Delta\mu^k. \tag{29}$$

We iterate until $\|u^{k+1} - u^k\| + |\mu^{k+1} - \mu^k| < \epsilon$.

If the iteration converges, we may repeat the procedure after the composite grid has been updated. The number of iterations which was required to get convergence is used to determine next stepsize $\Delta\sigma$. However, if the iteration diverges, we halve the stepsize and try again.

The continuation method requires the knowledge of the Jacobian matrix $\partial L/\partial u$ and the derivative $\partial L/\partial \mu$. Only the boundary conditions at the surface (18), (19) depend explicitly on μ. Hence, $\partial L/\partial \mu$ will only be nonzero along the surface. The Jacobian matrix is most naturally split into the two parts $\partial L/\partial \phi$ and $\partial L/\partial \eta$. L depends nonlinearly on ϕ only in (18) and (19), so $\partial L/\partial \phi$ is very straightforward to form. We refer the reader to [12] for a detailed description of how to set up the part $\partial L/\partial \eta$.

5.2. The singular case. The smallest eigenvalue, except for the shift-eigenvalue, is monitored along the solution curve to detect singular points. The pseudo-arclength method easily passes through limit points, but special care is necessary to switch branches at a bifurcation point. One can proceed as in [8]. In that method, the tangent to the solution curve on the other branch is computed by using second derivative information of L. The continuation then proceeds in the direction of the new tangent starting at the singular point. In the present work, we have instead used a Lyapunov–Schmidt reduction technique that does not use the second derivatives of L.

We only consider the situation when $\partial L/\partial u$ has exactly two small eigenvalues, one of which is the shift-eigenvalue. We further assume the eigenvectors of the two small eigenvalues to be linearly independent. Our task is to find one point on every solution curve (u,μ) that satisfies $(I-P[u,\mu])L[u,\mu]=0$, close to the singular point.

Let e'_r and e'_l be the right and left eigenvectors of the second small eigenvalue, normalized to have $\|e'_r\|=1$ and $\langle e'_l, e'_r\rangle = 1$. Let P' be the projection that maps $\mathcal{X}$ onto the second eigenspace. It is defined by $P'f = \langle e'_l, f\rangle e'_r$, $f\in\mathcal{X}$.

Let (u_0,μ_0) be a solution point close to the singular point and set $u=u_0+x$ and $\mu=\mu_0+\tau$. We split x into three parts, $x=x^I+x^{II}+x^{III}$, where $x^I=(I-P-P')x$, $x^{II}=P'x$, and $x^{III}=Px$. The part of x in P' can also be expressed as $x^{II}=\alpha e'_r$, $\alpha=\langle e'_l,x\rangle$, and the part in P will be set to zero, $x^{III}=0$, since it only shifts the solution. We now split the problem according to

$$(I-P')(I-P)L[u,\mu]=0, \tag{30}$$

$$P'(I-P)L[u,\mu]=0. \tag{31}$$

Let $A=\partial L/\partial u[u_0,\mu_0]$ denote the Jacobian matrix. We can solve the linear system $Ax=b$ for $b\in(I-P-P')\mathcal{X}$ uniquely in $(I-P-P')\mathcal{X}$, where P and P' are evaluated at (u_0,μ_0). Therefore, (30) can be used to compute $x^I=x^I(\alpha,\tau)$ by iteration. First, we choose some values of α and τ that will be fixed throughout the iteration. We take the initial guess to be $x_0^I=0$ and compute the subsequent iterates by solving

$$L_u[u_0,\mu_0]\Delta x_k = -(I-P-P')L[u_0+x_k^I+\alpha e'_r, \mu_0+\tau], \tag{32}$$

$$x_{k+1}^I = x_k^I + (I-P-P')\Delta x_k. \tag{33}$$

We truncate the iteration when $\|x_{k+1}^I - x_k^I\| < \epsilon$. The iteration will converge for sufficiently small α and τ because, by assumption, $\partial L/\partial u$ has only two small eigenvalues and the corresponding parts of the solution are kept constant by the projections. The iteration only requires L_u to be factored once; it is therefore relatively inexpensive compared to the Newton iteration.

The second relation (31) is equivalent to the scalar equation $g(\alpha,\tau)=0$, where

$$g(\alpha,\tau)=\langle e'_l, L[u_0+x^I(\alpha,\tau)+\alpha e'_r, \mu_0+\tau]\rangle. \tag{34}$$

This equation can, for instance, be solved by the following two-step technique. We first approximately locate the zeros of g by evaluating it at a number of points on the small circle $\alpha^2+\tau^2=r^2, r \ll 1$. For each zero, we solve $g=0$ by bisection with the approximate location as initial guess.

We can proceed with the nonsingular continuation approach starting from a solution point sufficiently far away from the singularity in the (α,τ)-plane. We trace out all solution curves that connect at the singular point by doing this for one point on each branch.

6. Changing the composite grid. When the surface moves during the continuation procedure, the amount of overlap between the subdomains Ω_Q and Ω_{Q-1} will change. Component grid Q has the highest priority so all its grid points will always be used in the composite grid. The situation is different for grid $Q-1$. After the surface has moved, some grid points that were previously used might no longer be needed, while some formerly unused grid points might be required in the composite grid. Therefore, the type of equation associated with a grid point and the number of used grid points in grid $Q-1$ will depend on the location of the surface. Furthermore, even if the type of equation remains unchanged for all grid points, the interpolation locations might be different for some interpolation points. For these reasons, the continuation method must be modified to allow for changes in the composite grid.

The basic idea is to fix the interpolation locations and the type of equation associated with each grid point during the Newton iteration and only update the mapping function for the surface grid. This means that, for instance, the discretization points are prevented from changing to interpolation points or unused points. For stability reasons, the amount the surface is allowed to move between two solution points is restricted by the amount of overlap, i.e., we do not allow for extrapolation in the interpolation relations. With the notation of (20), extrapolation occurs if at least one of the following inequalities is satisfied: $\tilde{r} < r^b_{j-1}$, $\tilde{s} < s^b_{k-1}$, $r^b_{j+1} < \tilde{r}$ or $s^b_{k+1} < \tilde{s}$. If extrapolation should occur in any interpolation relation, we stop the Newton iteration, decrease the stepsize $\Delta\sigma$ and restart the continuation from the previous solution point.

Once the iteration has converged to a solution where there is no extrapolation in the interpolation relations, we verify that the composite grid is consistent with the position of the surface. This is done by ensuring that the interpolations are sufficiently centered. The criterion

$$(35)\qquad r^b_j - 0.6h^b_r \le \tilde{r} \le r^b_j + 0.6h^b_r, \qquad s^b_k - 0.6h^b_s \le \tilde{s} \le s^b_k + 0.6h^b_s,$$

has been found to work well in practice. We have a valid solution if all interpolation points satisfy (35); we may then proceed with the continuation. Otherwise, the composite grid and the solution must first be corrected.

6.1. The correction step. We begin by constructing a corrected composite grid. We mention in passing that this can be done at a much lower computational cost than would be involved in constructing a completely new grid. Thereafter, we repeat the last continuation step on the corrected grid. We get u_0 on the corrected grid by interpolation from the previous grid. With this approach, it is possible to correct the grid close to a limit point. The simpler idea to solve $(I-P[u,\mu])L[u,\mu]=0$ on the corrected composite grid, but fixing μ at the current value, might fail close to a limit point because its location depends slightly on the grid (see Fig. 2). The correction step has been found to be very stable with the present restriction on the movement of the surface.

The reason for allowing the interpolation points to be slightly noncentered is to ensure that the corrected solution will satisfy (35) on the corrected grid. The difficulty which otherwise might arise is illustrated in Fig. 3. Consider the situation when 0.6 is

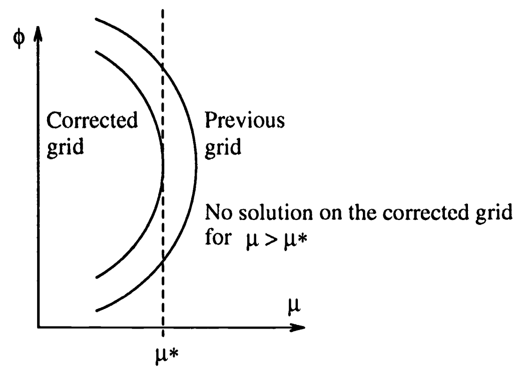


FIG. 2. *The location of a limit point might depend slightly on the grid.*

replaced by 0.5 in (35). Let some interpolation point have $\tilde{s} = s^b_k + 0.5h^b_s + \epsilon$, $0 < \epsilon \ll 1$ before the correction step. The corrected composite grid will use more grid points in grid b to make $\tilde{s} = s^b_{k+1} - 0.5h^b_s + \epsilon$. The problem is that the solution on the corrected grid might be slightly different than the solution on the previous grid. In particular, the corrected location of the surface can make $\tilde{s} = s^b_{k+1} - 0.5h^b_s - \tilde{\epsilon}$, $0 < \tilde{\epsilon} \ll 1$, which would call for correcting the composite grid back to the initial state. This problem does not occur if a less restrictive tolerance is used. Instead, both composite grids in Fig. 3 would be valid. The history of the previous locations of the surface determines which of the possible composite grids will be used. This is acceptable because we can expect the solutions on the different grids to be very similar if the solution is well resolved.

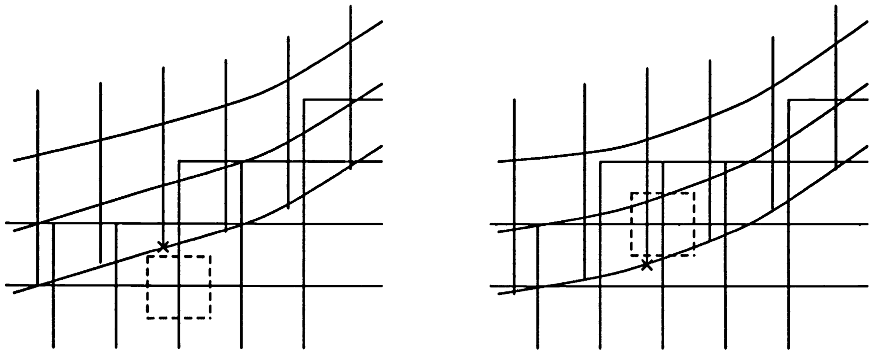


FIG. 3. *The oscillation between two very similar composite grids that can be avoided by allowing for slightly noncentered interpolation points.*

6.2. Adapting the grid to the solution. The solution becomes steeper as the wave height increases. In particular, the derivative of the surface elevation tends to a discontinuity at the crest of the wave. A very fine grid is therefore necessary to resolve the solution close to that point, but it is difficult to a priori estimate how fine the grid needs to be for a certain wave height. It is also very uneconomical to use the fine grid all the way from the trivial solution. For these reasons, we have developed an adaptive technique where we allow the resolution in each component grid and the stretching in the surface grid to change during the continuation procedure.

Ideally, one would like to monitor the truncation error of the discrete solution and increase the resolution when and where it is necessary. In the present work, we have used a simpler approach, namely, to monitor how well the solution is resolved on the grid. In particular, for the perturbation potential, we look at the largest difference in each direction, i.e.,

$$\text{(36)} \qquad \text{diff}_r\phi^q := \max|h_r^q D_{+j}\phi^q_{j,k}|, \qquad \text{diff}_s\phi^q := \max|h_s^q D_{+k}\phi^q_{j,k}|.$$

For the surface elevation, we are concerned with resolving the sharp gradient in η_x that develops when the wave height gets close to its limiting value (see Fig. 4). We therefore monitor $\text{diff}_r\eta_x$.

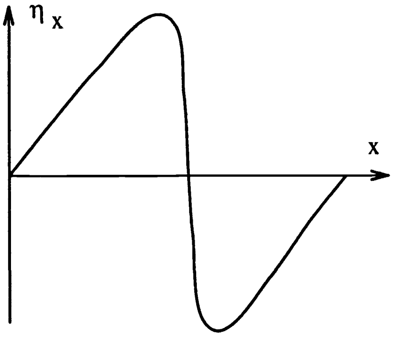


FIG. 4. *The x-derivative of the surface elevation close to the maximum wave height.*

At each valid solution point, we calculate $\text{diff}_r\phi^q$, $\text{diff}_s\phi^q$, and $\text{diff}_r\eta_x$. The idea is to keep these quantities in the predetermined ranges

$$\text{(37)} \qquad A_\phi < \text{diff}_r\phi^q < B_\phi, \quad A_\phi < \text{diff}_s\phi^q < B_\phi \quad \text{and} \quad A_\eta < \text{diff}_r\eta_x < B_\eta$$

by changing the number of grid points and the parameters in the stretching function. The constants A_ϕ, B_ϕ, A_η, and B_η will be called the resolution thresholds.

Initially, the solution is trivial, and we can use a coarse grid to start the continuation. During the continuation, some difference will eventually exceed its limit and we then need to construct a grid with better resolution. For example, let $\text{diff}_r\phi^q > B_\phi$. We increase the number of grid points in the r-direction of grid q by changing N_q such that the solution on the new grid approximately satisfies $\text{diff}_r\phi^q = A_\phi$.

We control the stretching function (10) by monitoring $\eta_x(x)$ (see Fig. 4). To avoid wasting grid points, we adjust the parameters β_r and γ_r to make $h_r^Q D_{+j}\eta_x(x_j^Q)$ as uniform over the grid as possible. It is clear that $|\eta_{xx}|$ has maxima at the crest ($x = x_{\text{crest}}$) and at the trough ($x = x_{\text{trough}}$) of the wave. To concentrate the layer in the stretching function around the crest, we choose $\gamma_r = x_{\text{crest}}/l$. The ratio between the largest and the smallest grid sizes equals approximately β_r. We therefore take $\beta_r = |\eta_{xx}(x_{\text{crest}})/\eta_{xx}(x_{\text{trough}})|$.

When the number of grid points and the stretching have been properly adjusted, we perform a correction step (§6.1) to update the grid and get a valid solution on the refined grid. To make the process stable and reliable, the solution must be reasonably well resolved on the grid before we try to change the resolution. If it is not, we decrease the stepsize $\Delta\sigma$ and compute an intermediate, more well-resolved, solution before we change the resolution. Without this precaution, the correction step might diverge. Another important rule is to avoid changing the resolution too drastically. Practically, we found that the correction step converges quickly if the number of grid points changes by less than 20% in each direction on each component grid, and the stretching strength β_r changes by less than an absolute amout of 5.0.

7. Implementation. In the present implementation we have used the software package CMPGRD (cf. [1]) to generate the composite grids. A composite grid is created by calling CMPGRD as a subroutine with an input consisting of the component grid transformations, the number of grid points in each direction on each grid, and the priority between the component grids. On output, each grid point is labeled according to how it will be used and the interpolation locations are also given for the interpolation points.

When CMPGRD is called to correct an existing grid, the previous composite grid and the corrected mapping function for the surface grid are given as input. For the case when the resolution has been changed, the new number of grid points is also supplied. The fast algorithm for correcting the composite grid only works if the new interpolation points can be found, at most, one grid point away from some previous interpolation point in the same component of the previous composite grid (cf. Fig. 5). This restriction is consistent with the aforementioned requirement to disallow for extrapolation in the interpolation relations. However, we have encountered situations when the fast algorithm fails after the stretching has been changed drastically. In those rare cases, a new composite grid must be constructed from scratch.

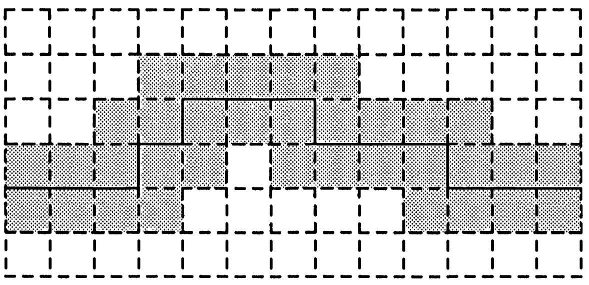


FIG. 5. *The band in the (r, s)-plane that is searched for the location of the interpolation points in the corrected grid. The solid line indicates the position of the interpolation points in the previous grid.*

The information supplied by CMPGRD is sufficient to form the discrete set of equations for a composite grid that is composed of an arbitrary number of component grids, each having an arbitrary number of grid points. The only restriction on the component grids is that they must overlap sufficiently. However, the implementation of a general solver in FORTRAN-77 requires a dynamic memory allocator. For this purpose, we have used the DSK-package, which is an autonomous part of CMPGRD.

The emerging linear systems of equations were solved by the YALE sparse matrix package (cf. [6]). We found experimentally that the accuracy of the solution was improved when the equations corresponding to the boundary conditions at the surface were inserted early in the system matrix. Actually, this trick was necessary to make the Newton iteration converge properly for solutions with wave heights close to the limiting value. The reason for this behavior might be that this package does not pivot for numerical stability when it LU-decomposes the matrix. Our main motivation for using this package despite this deficiency is that it is fast.

8. Numerical results. We begin by studying a single wave, i.e., $l = 2\pi$, on deep water. We compare our results with those reported by [2] and [10] for the infinitely deep case. To investigate the effect of the finite depth in our computation, we performed two sets of calculations with $d = 2\pi$ and $d = 4\pi$, respectively. To also investigate the dependence of the grid size, we varied the resolution thresholds. For simplicity, we used $A_\phi = A_\eta$ and $B_\phi = B_\eta = A_\phi + 5\times10^{-3}$. We used a composite grid with four components and the mapping functions for the component grids had the parameters $a_1 = -d$, $b_1 = -2.5$, $a_2 = -3.0$, $b_2 = -0.7$, $a_3 = -1.2$, $b_3 = 0.65$, and $b_4 = 0.4$. In the case $l = d$, we used the following number of grid points in the start grid: $N_1 = 14$, $M_1 = 8$, $N_2 = 23$, $M_2 = 11$, $N_3 = 39$, $M_3 = 13$, $N_4 = 63$, and $M_4 = 7$. For the deeper case, we used $M_1 = 20$.

For the continuous problem, the bifurcation from the trivial solution occurs at $\mu_0 = \sqrt{\tanh d}$. With the present start grid, it was displaced by $\mathcal{O}(10^{-4})$.

Let h denote the wave height. In Tables 1 and 2 we present h/l as function of the resolution thresholds. The three different μ-values correspond to solutions where the maximum slope of the surface is approximately 17.0, 22.4, and 27.8 degrees, respectively. For $A_\phi = 10^{-2}$, the absolute difference between our results and those reported by [10] are of the order $\mathcal{O}(10^{-3})$. We conjecture that the discrepancies are mainly caused by truncation errors and not by the finite depth. Both the accuracy and the efficiency of the present method would probably be improved by using a higher-order difference scheme.

TABLE 1
h/l as function of the resolution for the depth $d = 2\pi$. The results obtained by [10] *for infinite depth are given at the bottom of the table.*

A_ϕ	$\mu = 1.04247$	$\mu = 1.07029$	$\mu = 1.09184$
3.0×10^{-2}	0.092107	0.117823	0.135427
2.0×10^{-2}	0.092481	0.118119	0.136440
1.0×10^{-2}	0.092278	0.117885	0.136362
[10]	0.091809	0.117572	0.136178

TABLE 2
h/l as function of the resolution for the depth $d = 4\pi$.

A_ϕ	$\mu = 1.04247$	$\mu = 1.07029$	$\mu = 1.09184$
3.0×10^{-2}	0.092083	0.117908	0.135623
2.0×10^{-2}	0.092557	0.118228	0.136455
1.0×10^{-2}	0.092257	0.117877	0.136353

The value of $\phi(0, -d)$, which occurs in (21), indicates how well the conservation property of the continuous problem is satisfied. By starting from the trivial solution, it is initially zero. For $A_\phi = 10^{-2}$ it was of the order $\mathcal{O}(10^{-6})$ along the solution curve.

To demonstrate the result of the adaptation technique, we present the start grid in Fig. 6 and the grid corresponding to $A_\phi = 3\times10^{-2}$, $\mu = 1.09184$ in Fig. 7.

In Figs. 8 and 9, we plot h/l as function of the phase velocity for the depth $d = l$ and the resolution $A_\phi = 10^{-2}$. During the computation of this solution curve, the stepsize in the pseudo-arclength was chosen to make the Newton iteration converge in approximately four steps. We calculated 43 solution points and the resolution was changed 24 times. The composite grid was corrected 43 times due to noncentered interpolations; six of those were done after the resolution had been changed. The fast algorithm for correcting the composite grid did not fail during any correction step. This implies that the

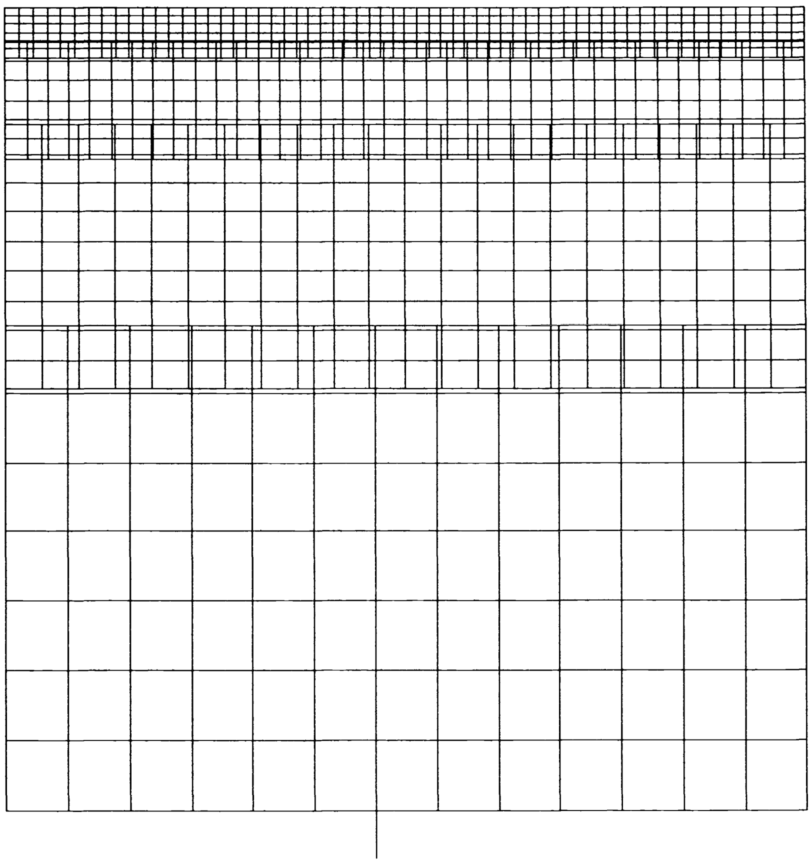


FIG. 6. *The start grid.*

overhead for changing the resolution corresponded to 30 out of a total of 110 continuation steps, i.e., approximately 27%. To give an example of the amount of work that was saved by using an adaptive grid, we report the number of equations as function of h/l in Fig. 10. The sparse matrix solver requires of the order $\mathcal{O}(n^2)$ operations to perform one Newton step. This means that one Newton iteration with 10^4 grid points requires the same effort as performing $\mathcal{O}(100)$ iterations with 10^3 grid points. These figures clearly show the benefit of using an adaptive grid.

Next, we investigate the subharmonic bifurcation reported by [2] for the infinitely deep case. In this computation, we started with two waves in the domain $l = 4\pi$. The depth was $d = 2\pi$ and the grid sizes for the start grid were the same as in the single-wave case. We used the resolution threshold $A_\phi = 10^{-2}$. The smallest eigenvalue except for the shift-eigenvalue was monitored along the solution curve to detect singular points. Initially, the solution has period $l/2$ and it is identical to two adjacent single wave solutions. A solution with this property will be called regular.

In close agreement with [2], we found a singular point in the vicinity of $\mu = 1.08414$, $h/l = 0.06447$. By applying the technique described in §5.2, we detected a bifurcating solution curve that connects to the regular solution curve at the singular point. The eigenfunction corresponding to the second small eigenvalue is given in Fig. 11. It has period l in the x-direction, which implies that the solution on the bifurcated curve also has

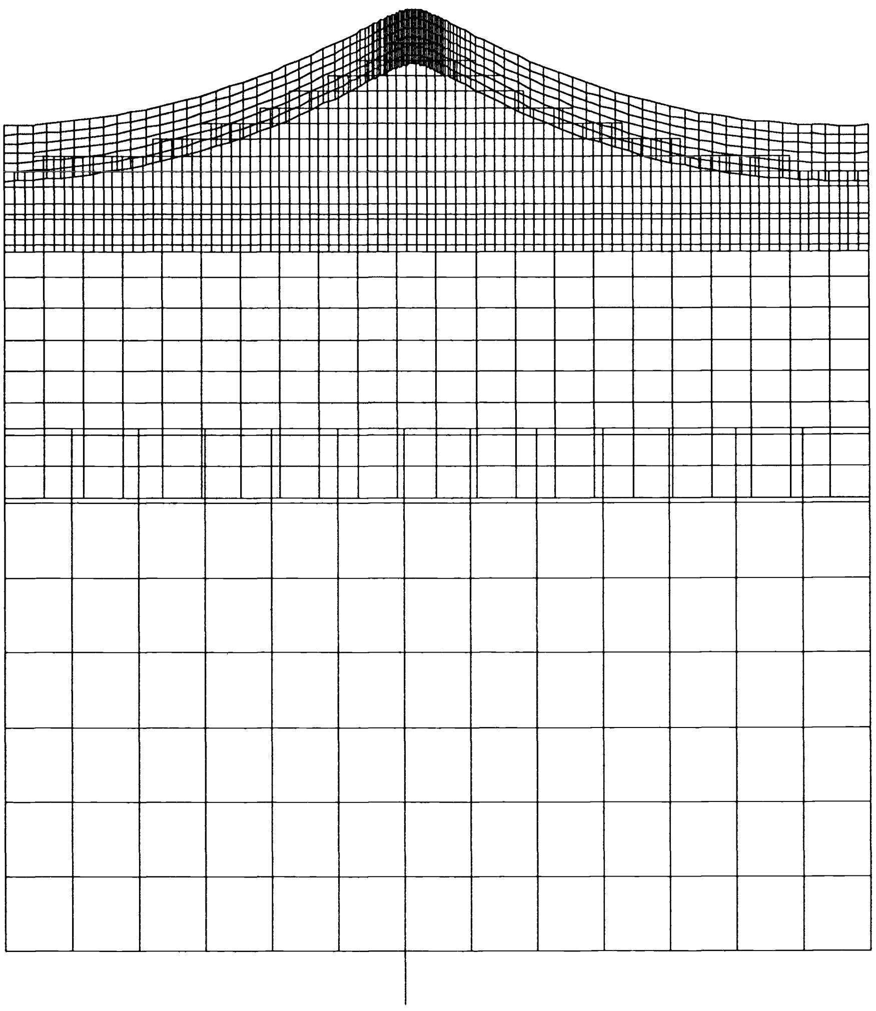


FIG. 7. *The grid at* $\mu = 1.09184$ *with the resolution* $A_\phi = 3 \times 10^{-2}$.

period l, i.e., twice the period of the regular solution. The bifurcated solution curve was traced out with the nonsingular continuation method starting on the bifurcated branch close to the singular point. On the bifurcated solution curve, one of the crests becomes sharp while the other stays rounded when the wave height increases (see Fig. 12). Obviously, the two possible locations of the sharp crest correspond to identical solutions shifted by $l/2$; they are found by proceeding in opposite directions along the solution curve. In Fig. 13, we give h/l as function of the phase velocity for the bifurcated solution. The continuation was truncated when the number of equations exceeded 25,000.

A direct comparison with previous results for the shallow water case [4], [14] was not possible because those results were obtained by keeping the flux μd constant during the continuation. In the present method, we instead kept the constant in the Bernoulli equation (3) fixed, which implies that the flux varies slightly along our solution curves. The average depth in the two approaches will therefore be different at the same velocity for nonzero wave heights. However, an approximate comparision was performed by interpolating between the results in [4] for different depths. We made the comparison

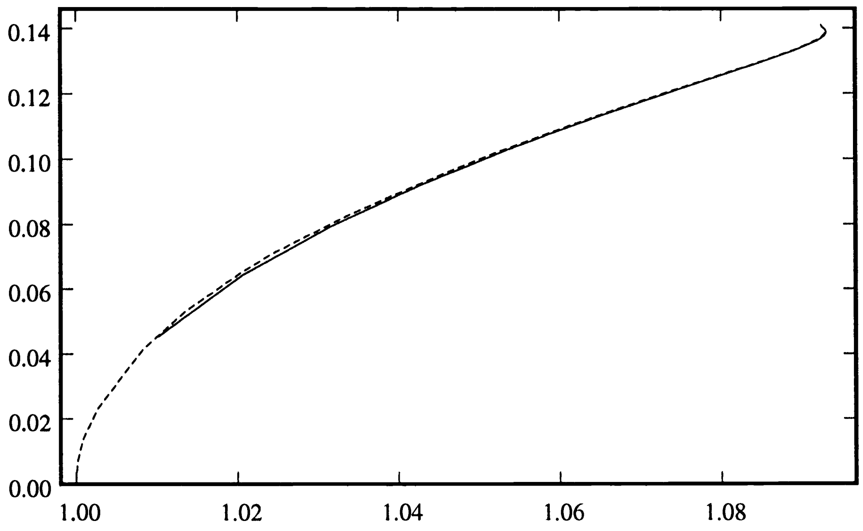


FIG. 8. *h/l as function of the phase velocity for the case $d = 2\pi$ and $A_\phi = 10^{-2}$. The solid line represents the result reported by* [10]; *the dashed line corresponds to the present work.*

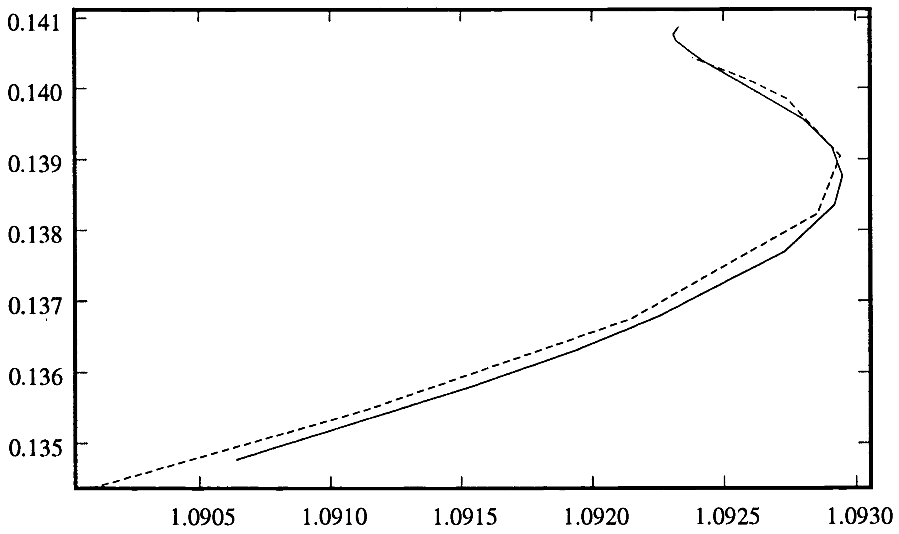


FIG. 9. *The same case as in Fig.* 8, *close to the solution of maximum height. Here, the solid line corresponds to the results of* [2].

for the solution curve with the initial depth $d = 1.2039728$, which corresponds to Table A3 in [4]. In our computation, we used the resolution threshold $A_\phi = 10^{-2}$ and traced out the solution until the number of gridpoints exceeded 15,000. The average depth

$$\bar{d} = \frac{1}{l}\int_0^l (\eta(\xi) + d)\, d\xi$$

was computed along the the present solution curve to make a comparison with Cokelet's results possible. The difference in depth did not exceed 5% between the solution curves, and the average depth in the present calculation was always smaller than those in Table

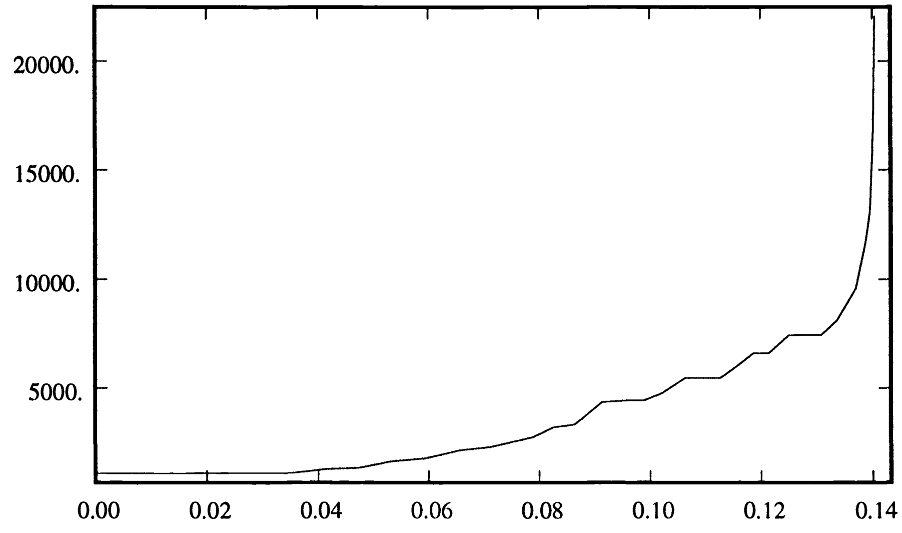


FIG. 10. *The number of equations as function of h/l for the case $d = 2\pi$ and $A_\phi = 10^{-2}$. The first grid had* 1,086 *points and the last grid had* 22,053 *points.*

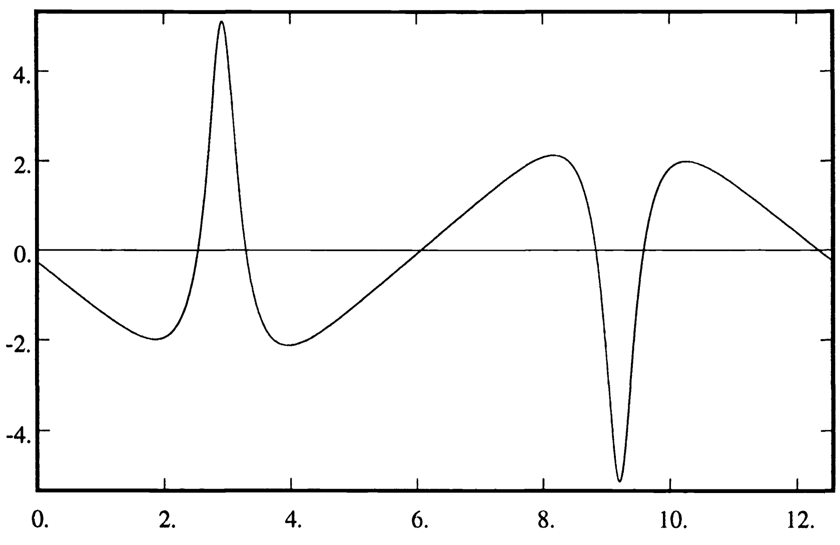


FIG. 11. *The surface component of the eigenfunction connected to the second small eigenvalue, close to the bifurcation point.*

A3. In order to interpolate, we also used the data for the initial depths $d = 1.6094379$ and $d = 0.91629073$, i.e., Tables A2 and A4 in [4], respectively. For each solution curve, a scaled phase velocity was defined by $c = (\mu - \min \mu)/(\max \mu - \min \mu)$. Let the data points in the present computation be denoted c_i, $\bar{d}_i$, and h_i, $i = 1, 2, \dots, n_p$, and let those corresponding to Table Ak be denoted $c_i^{(k)}$, $\bar{d}_i^{(k)}$, and $h_i^{(k)}$, $i = 1, 2, \dots, n_c$. The following interpolation procedure was applied: A cubic spline parametrized by the index was interpolated through the data points in each of Cokelet's tables: $c_{\text{int}}^{(k)}(\xi)$, $\bar{d}_{\text{int}}^{(k)}(\xi)$, and $h_{\text{int}}^{(k)}(\xi)$, $k = 2, 3, 4$. For each $i = 1, 2, \dots, n_p$, we solved $\xi_{ki} : c_{\text{int}}^{(k)}(\xi_{ki}) = c_i$ and evalu-

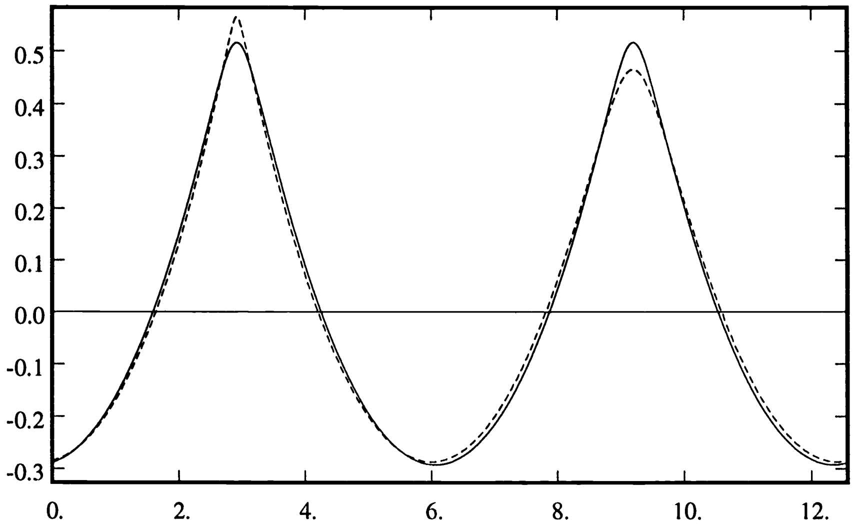


FIG. 12. *The surface elevation as function of x. The solid line represents the regular solution close to the bifurcation point* ($\mu = 1.08414$, $h/l = 0.06447$) *and the dashed line corresponds to the bifurcated solution at* $\mu = 1.08132$, $h/l = 0.06786$.

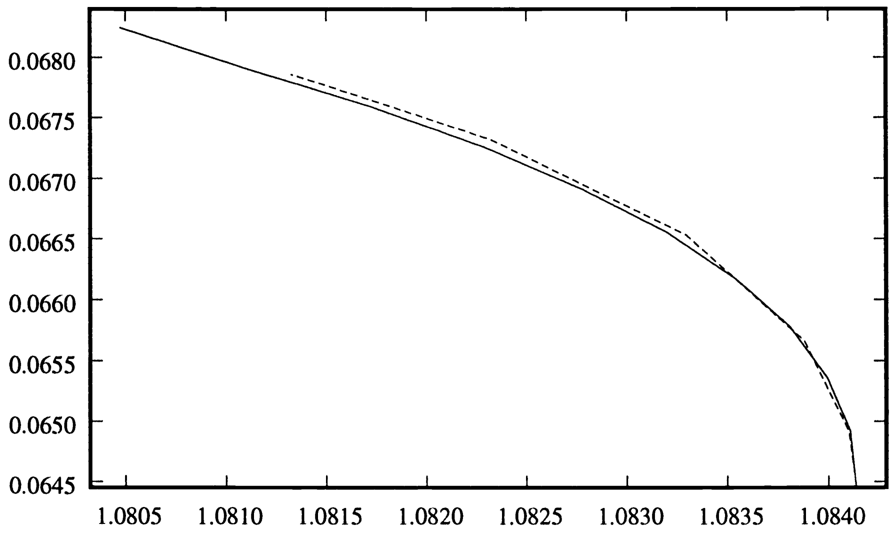


FIG. 13. *h/l as function of the phase velocity for the bifurcated solution. Here, $d = 2\pi$ and $A_\phi = 10^{-2}$. The solid line represents the infinitely deep case reported by* [2] *and the dashed line corresponds to the present work.*

ated $h_{\mathrm{int}}^{(k)}(\xi_{ki})$ and $\bar{d}^{(k)}(\xi_{ki})$. We then assumed the interpolated average depth to behave like

$$(38)\quad \bar{d}(\eta) = \frac{(\eta-1)(\eta-2)}{2}\bar{d}^{(2)}(\xi_{2i}) - \eta(\eta-2)\bar{d}^{(3)}(\xi_{3i}) + \frac{\eta(\eta-1)}{2}\bar{d}^{(4)}(\xi_{4i}).$$

Thereafter, (38) was solved for $\eta_i : \bar{d}(\eta_i) = \bar{d}_i$. The wave height was also assumed to vary parabolically for each fixed, i.e.,

$$(39) \quad h(\eta) = \frac{(\eta-1)(\eta-2)}{2} h^{(2)}(\xi_{2i}) - \eta(\eta-2)h^{(3)}(\xi_{3i}) + \frac{\eta(\eta-1)}{2} h^{(4)}(\xi_{4i}).$$

The interpolated wave height was found by evaluating (39) for η_i. The results are given in Fig. 14. The absolute difference in relative wave height was of the order $\mathcal{O}(10^{-2})$. The discrepancy between the results is one order of magnitude larger for the shallow water case compared to the deep water case. We conjecture that this is related to the interpolation error.

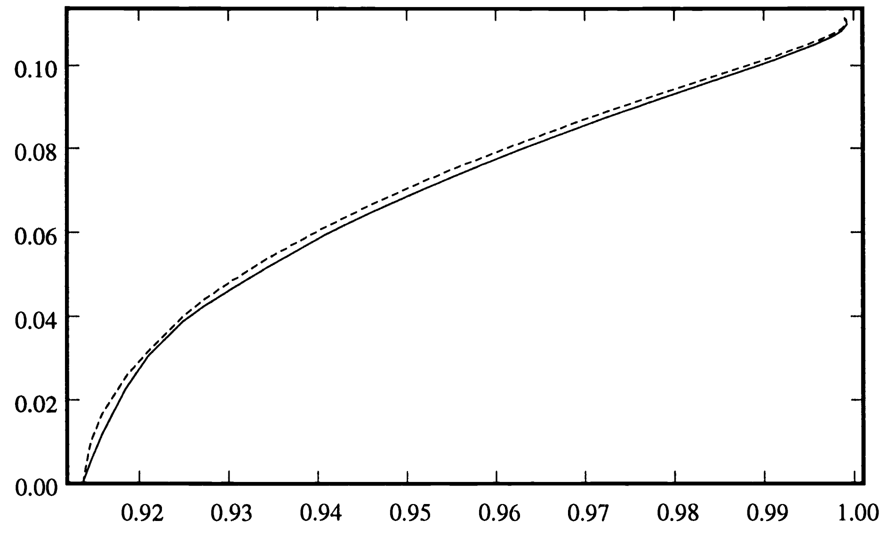


FIG. 14. *h/l as function of the phase velocity for the case $d = 1.2039728$ and $A_\phi = 10^{-2}$. The solid line represents results interpolated from data reported by* [4] *and the dashed line corresponds to the present work.*

9. Conclusions. In this paper, it has been shown that composite overlapping grids together with finite difference methods can be used to accurately calculate steep periodic water waves. It has been indicated that an adaptive grid is necessary to achieve well-resolved solutions close to the wave of maximum height. We have shown that the overhead connected with changing the grid is small compared to the cost involved in always using a fine grid.

An underwater obstacle could easily be introduced by adding a component grid close to the obstacle and replacing the periodicity in the x-direction with appropriate in- and out-flow boundary conditions. The solution is only asymptotically periodic behind the obstacle, and it would be interesting to see if the subharmonic bifurcation also appears here. This will be investigated in a future paper.

Acknowledgment. Professor H. -O. Kreiss is gratefully acknowledged for many inspiring discussions on the present problem. I am indebted to Dr. G. Chesshire who designed and implemented the algorithm for effectively constructing a corrected composite grid. He and Dr. D. L. Brown are also thanked for sharing their thorough knowledge on composite overlapping grids.

REFERENCES

[1] D. L. Brown, G. Chesshire, and W. D. Henshaw, *Getting Started with CMPGRD, Introductory User's Guide and Reference Manual,* LA–UR–90–3729, Los Alamos National Laboratory, Los Alamos, NM, manuscript.

[2] B. Chen and P. G. Saffman, *Numerical evidence for the existence of new types of gravity waves of permanent form on deep water*, Stud. Appl. Math., 62 (1980), pp. 1–22.

[3] G. Chesshire and W. D. Henshaw, *Composite overlapping meshes for the solution of partial differential equations,* J. Comput. Phys., 90 (1990), pp. 1–63.

[4] E. D. Cokelet, *Steep gravity waves in water of arbitrary uniform depth,* Philos. Trans. Roy. Soc. London Ser. A, 286 (1977), pp. 183–230.

[5] F. Dias and J.-M. Vanden-Broeck, *Open channel flows with submerged obstructions,* J. Fluid Mech., 206 (1989), pp. 155–170.

[6] S. Eisenstat, M. Gursky, M. Shultz, and A. H. Sherman, *The Yale Matrix Package* II: *The Non-Symmetric Case*, Rep. 114, Dept. of Computer Science, Yale University, New Haven, CT, manuscript.

[7] A. D. Jepson and H. B. Keller, *Steady State and Periodic Solution Paths: their Bifurcations and Computations,* Bifurcation: Analysis, Algorithms and Applications, T. Kupper, H. D. Mittelmann, and H. Weber, eds., Birkhauser ISNM series, No. 70, 1984, pp. 219–246.

[8] H. B. Keller, *Numerical Solution of Bifurcation and Nonlinear Eigenvalue Problems,* in Applications of Bifurcation Theory, P. H. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359–384.

[9] Y. H.Kim and T. R. Lucas, *Nonlinear Ship Waves*, Proc. 18th Symp. Naval Hydrodynamics, Ann Arbor, MI, 1990.

[10] M. S. Longuet-Higgins, *Integral properties of periodic gravity waves of finite amplitude*, Proc. Roy. Soc. London Ser. A, 342 (1974), pp. 157–174.

[11] N. A. Petersson and J. F. Malmliden, *Computing the flow around a submerged body using composite grids,* J. Comput. Phys., 105 (1993), pp. 47–57.

[12] N. A. Petersson, *Computing Periodic Gravity Waves on Water by Using Moving Composite Overlapping Grids,* CAM Report 92-01, Dept. of Mathematics, University of California, Los Angeles, 1992.

[13] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin, *Numerical Grid Generation,* North-Holland, Amsterdam, 1985, p. 126.

[14] J.-M. Vanden-Broeck and L. W. Schwartz, *Numerical computation of steep gravity waves in shallow water,* Phys. Fluids, 22 (1979), pp. 1868–1871.

[15] G. B. Whitman, *Linear and Nonlinear Waves*, Wiley-Interscience, New York, 1974, p. 434.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1359–1367, November 1993

A MODIFIED BROYDEN UPDATE WITH INTERPOLATION*

MIGUEL F. ANJOS†

Abstract. A new class of Hessian updates is introduced for quasi-Newton methods obtained by modifying the Broyden class of updates so that it interpolates function values. In general, the members of this new class do not satisfy the quasi-Newton equation. Through the use of a weak inverse updating strategy, a specific choice of update within this class is then introduced. Numerical results are given to illustrate the behaviour of this new update and to compare it with other updating sequences.

Key words. unconstrained optimization, quasi-Newton methods, Broyden class, interpolation, inverse sizing, weak sizing, nonlinear optimization

AMS subject classifications. 65K, 90C, 49M

1. Introduction. This paper considers quasi-Newton methods for solving

$$\min_{x \in \Re^n} f(x), \tag{1.1}$$

where f is assumed to be twice continuously differentiable. However, only first derivative (gradient vector) evaluations are required by these methods.

These methods are iterative. The required initial data is a starting point $x_0 \in \Re^n$ and an initial approximation B_0 for the Hessian $\frac{\partial^2 f}{\partial x^2} \in \Re^{n \times n}$ of f, which is typically chosen as $B_0 = I_{n \times n}$, when no additional information about f is known a priori.

Given this initial information, at the kth iteration, a local quadratic model $\overline{f}$ is used to approximate f

$$\overline{f}(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T B_k s, \tag{1.2}$$

where $g_k \equiv \nabla f(x_k)$, and B_k is the current approximation to the Hessian. Minimizing $\overline{f}$ gives the descent direction

$$p = -B_k^{-1} g_k,$$

which is used for f in a line search satisfying the Wolfe conditions (see [1] and [3] for more details). A new point x_{k+1} is computed and then taken as the next point for the iteration.

A key element of each iteration is the updating of the B_k matrix by taking advantage of the new information about f gathered during the current iteration. Usual requirements are that the update B_{k+1} must be symmetric positive definite and satisfy the secant or quasi-Newton equation

*Received by the editors February 12, 1992; accepted for publication (in revised form) November 12, 1992. This research was partially supported by Natural Sciences and Engineering Research Council Summer Undergraduate Research Award.

†School of Computer Science, McGill University, Montreal, Quebec, Canada, H3A 2A7. Present address: Scientific Computing and Computational Mathematics, Building 460, Stanford University, Stanford, California 94305-2140 (na.anjos@na-net.ornl.gov).

$$B_{k+1}\, s = y, \tag{1.3}$$

where $s \equiv x_{k+1} - x_k$ and $y \equiv g_{k+1} - g_k$. This equation effectively requires that B_{k+1} approximate the curvature of f along the step s. The main motivation for imposing that B_{k+1} satisfies (1.3) follows from the fact that if the real Hessian were constant (i.e., if f were a quadratic function), then after n quasi-Newton linearly independent steps, B_n would equal the real Hessian [4].

One class of updates satisfying these conditions is the Broyden family of updates:

$$B_{k+1} = B_k - \frac{B_k\, s\, s^T B_k}{s^T B_k\, s} + \frac{y\, y^T}{y^T\, s} + (1-\phi) s^T B_k\, s\, w\, w^T, \tag{1.4}$$

where $y^T\, s > 0$, B_k is symmetric positive definite, and

$$w \equiv \frac{y}{y^T\, s} - \frac{B_k\, s}{s^T B_k\, s}.$$

Since B_{k+1} positive definite is required, an additional condition is that $\phi < \frac{ac}{ac-b^2}$, where we have

$$a = y^T B_k^{-1} y, \quad b = y^T s \quad \text{and} \quad c = s^T B_k\, s.$$

For $\phi = 1$ this becomes the BFGS update; for $\phi = 0$ the DFP update is obtained.

Corresponding update formulas for B_k^{-1} can also be used. For more details, refer to [1]–[3].

2. Recent developments. In this section we introduce two updating techniques that have been developed recently. The first is a Broyden class update which incorporates sizing techniques. The second is a BFGS update modified to satisfy an interpolation condition used in the study of conic methods for unconstrained optimization.

2.1. The Dennis–Wolkowicz update. The research and extensive testing of Dennis and Wolkowicz in [2] yielded a Broyden class update that improves on the BFGS update. This update possesses a self-scaling property as we now describe. Scaling is presented by Oren and Luenberger in [6] as a way to improve the performance of quasi-Newton updates. Following [2], we shall term it *sizing*.

Sizing consists in replacing at each iteration the current B_k by a scalar multiple of it and subsequently applying an updating formula. Both B_k and B_k^{-1} can be sized and the results are generally different. Sizing the latter is termed inverse sizing and we only discuss this technique.

Inverse sizing is effective in the first iteration of a quasi-Newton algorithm, but in subsequent iterations it may ruin the spectral information previously gathered in B_k. Dennis and Wolkowicz [2] observe that inverse sizing

$$\tilde{B}_k \leftarrow \frac{y^T B_k^{-1} y}{y^T\, s} B_k \tag{2.1}$$

effectively yields a matrix for which the condition

$$y^T \tilde{B}_k^{-1} y = y^T s \tag{2.2}$$

holds. They thus suggest that while this condition is satisfied, the change in the spectrum of B_k should be minimized. They then show that the rank-one update of B_k^{-1}

$$\overline{B}_k^{-1} = B_k^{-1} + \frac{1}{(y^T B_k^{-1} y)^2} (y^T s - y^T B_k^{-1} y) B_k^{-1} y y^T B_k^{-1} \tag{2.3}$$

is the unique solution to

$$\min_{\tilde{B}} \| W^T (\tilde{B}^{-1} - B_k^{-1}) W \|_F, \tag{2.4}$$

$$\text{s.t.} \qquad y^T \tilde{B}^{-1} y = y^T s \tag{2.5}$$

for any W such that $B_k = W W^T$. In this sense, $\overline{B}_k^{-1}$ is the closest matrix to B_k^{-1} satisfying (2.2). We note also that $\overline{B}_k^{-1}$ is symmetric positive definite if and only if $y^T s > 0$. The use of the update (2.3) to obtain a matrix satisfying (2.2) is termed *weak inverse updating*. Refer to [2] for more details.

The Dennis–Wolkowicz update performs weak inverse updating with the rank-one update (2.3) followed by a BFGS update of $\overline{B}_k^{-1}$. Interestingly, this sequence of two inverse updates is equivalent to a single rank-two update of B_k. This update is performed as follows: let

$$a = y^T B_k^{-1} y, \quad b = y^T s \quad \text{and} \quad c = s^T B_k s.$$

1. *First iteration*: Inverse size the current Hessian approximation, i.e.,

$$B_0 \leftarrow \frac{a}{b} B_0$$

and perform a BFGS ($\phi = 1$ in (1.4)) update on the sized matrix. Inverse sizing is done explicitly to avoid possible ill conditioning.

2. *Subsequent iterations*: Use the Broyden class update with

$$\phi = \phi^* \equiv \frac{1}{\frac{b}{c} - \frac{b^2}{ac} + 1},$$

which is equivalent to weak inverse updating followed by BFGS [2, Thm. 4.5 (iv)].

This update is hereditarily positive definite and it satisfies the quasi-Newton equation (1.3).

2.2. The Yuan update. The modified BFGS update introduced by Yuan in [8]

$$B_{k+1} = B_k - \frac{B_k\, s\, s^T\, B_k}{s^T\, B_k\, s} + t_k\, \frac{y\, y^T}{y^T\, s} \tag{2.6}$$

is a modification of the BFGS update arising from considering the interpolation condition

$$s^T\, B_{k+1}\, s = \alpha \equiv 2\,[f(x_k) - f(x_{k+1}) + s^T\, g_{k+1}], \tag{2.7}$$

which is used in Davidon's study of conic models, as summarized in [7]. This condition is equivalent to requiring function values from the previous iteration to be matched by the conic model. It is motivated by considering that the use of function values as well as gradient values should increase the efficiency of the overall optimization algorithm [7].

While such conic models can satisfy both the quasi-Newton equation (1.3) and (2.7), this is not possible with the quadratic model $\overline{f}$. However, Yuan [8] shows that condition (2.7) will be satisfied by B_{k+1} in (2.6) if we let

$$t_k = \frac{\alpha}{y^T\, s},$$

which is then truncated if necessary to lie in the interval

$$0.01 \le t_k \le 100.$$

Yuan [8] argues that this truncation ensures global convergence for convex f with the inexact line searches typically used by these methods. Furthermore, B_{k+1} is symmetric positive definite if and only if $t_k > 0$ together with B_k symmetric positive definite and $b > 0$. This update is not a member of the Broyden class (1.4) and, in general, it does not satisfy the quasi-Newton equation (1.3).

3. The new MBI update. Motivated by wanting to exploit the information available in known function values, the interpolation (2.7) contains valuable new information about f at each iteration. We thus consider imposing it on the entire Broyden class. For B_{k+1} a member of the Broyden class (1.4), we have

$$\begin{aligned} s^T\, B_{k+1}\, s &= s^T\, B_k\, s - s^T\, B_k\, s + s^T\, y + (1-\phi)s^T\, B_k\, ss^T\, ww^T\, s \\ &= s^T\, y \end{aligned}$$

since it is easily verified that

$$w^T\, s = s^T\, w = 0,$$

which implies that the last term of the Broyden class of updates has no influence on whether or not the interpolation condition is satisfied. Making the same minor modification to its general member as the modification made in [8] will yield a class of updates satisfying the interpolation condition (2.7), namely,

$$B_{k+1} = B_k - \frac{B_k\, s\, s^T B_k}{s^T B_k\, s} + t_k \frac{y\, y^T}{y^T s} + (1-\phi) s^T B_k\, s\, w\, w^T, \tag{3.1}$$

where $t_k = \frac{\alpha}{y^T s}$ and truncated to $0.01 \le t_k \le 100$ for the same reasons as in [8].

This new class is the basis for defining a new Hessian update, which we shall denote by MBI (modified Broyden update satifying the interpolation condition), and into which we incorporate the weak inverse updating properties of the Dennis–Wolkowicz update via the ϕ parameter.

The MBI update is performed as follows: Again let $a = y^T B_k^{-1} y$, $b = y^T s$, and $c = s^T B_k s$;

1. *First iteration*: Inverse size the current Hessian approximation

$$B_0 \leftarrow \frac{a}{b} B_0$$

to avoid possible ill conditioning and perform the $\phi = 1$ update of the class (3.1).

2. *Subsequent iterations*: Use the update of the class (3.1) with

$$\phi = \phi^* \equiv \frac{1}{\frac{b}{c} - \frac{b^2}{ac} + 1}.$$

Positive definiteness of the MBI update follows from $t_k > 0$ and the hereditary positive definiteness of the Dennis–Wolkowicz update.

It is important to note that the MBI update will not generally satisfy the quasi-Newton equation (1.3). Furthermore, because of the choice $\phi = \phi^*$, it is not simple to extend one of the classical convergence arguments in quasi-Newton methods to the MBI update.

However, the quasi-Newton equation and the interpolation (2.7) will be simultaneously satisfied when

$$t_k = \frac{\alpha}{y^T s} = 1$$

and the MBI update is then a member of the Broyden class. Standard local Q-superlinear convergence results should then be extendable to the MBI update. Our numerical tests have shown that this seems to correspond to the asymptotic behaviour of the MBI update. To illustrate this fact, we include in the numerical results section an example of the behaviour of t_k for the MBI update when solving the Wood function problem. Thus, as in [8], we believe it is reasonable to expect local Q-superlinear convergence for a quasi-Newton algorithm using the MBI updating formula.

4. Numerical results. The update MBI was tested for comparison with the $\phi = 1$ Broyden update (BFGS), the Dennis–Wolkowicz update (DW) and the Yuan modified BFGS update (Yuan). We also included two other updating sequences, namely the BFGS and Yuan updates with inverse sizing at the first iteration only (BFGS+IS and Yuan+IS, respectively).

The tests were performed on a DECstation 3100 using the same MATLAB code as in [2] that implements the routines described in [1]. The algorithm followed the minimization procedure for quasi-Newton methods described in §1 of this paper. In particular, a failure to converge for a method was declared after 450 unsuccessful iterations. At each step of the algorithm, a line search (Algorithm LINESEARCHMOD in [1]) was used to find a step satisfying the Wolfe conditions. Finally, we note that the criteria used to declare that a minimum of f has been reached is developed and commented on thoroughly in [1]. We thus refer the reader to that reference for further details on the algorithm.

We include the results for the six updating sequences on six test problems from [5]. For each problem three different starting points were considered:

- the suggested starting point in [5];
- the suggested starting point scaled by 10;
- the suggested starting point scaled by 100.

We tested the updates on all the problems from [5] and report a sample of these results which suggests the types of problems for which the MBI update appears to offer an improvement over the other methods. They also reveal instances where the MBI behaves rather poorly.

In the listings, F denotes a failure to solve the problem after 450 iterations. The data is reported in the format *number of iterations/number of function evaluations*.

The first problem shows that as the scaling increases, the sized updates are performing much better than the other two updates. It is not simple to determine a dominating update here, though.

Problem 1. *Wood function* (*dimension* $= 4$). Starting point: $(-3, -1, -3, -1)$. Exact minimizer: $(1, 1, 1, 1)$.

Scaling	BFGS	DW	Yuan	BFGS+IS	Yuan+IS	MBI
1	40/44	37/41	36/38	35/39	38/40	36/38
10	100/107	62/65	101/108	67/69	64/67	64/66
100	324/333	105/110	324/326	104/110	107/116	106/115

The same difference between sized and nonsized updates is observed in the next problem. It is possible though to observe a difference in performance between the updates. We note that the BFGS+IS and Yuan+IS updates dominate while the MBI performs rather poorly.

Problem 2. *Extended Powell function* (*dimension* $= 16$). Starting point: $(3, -1, 0, 1, \ldots, 3, -1, 0, 1)$. Exact minimizer: $(0, \ldots, 0)$.

Scaling	BFGS	DW	Yuan	BFGS+IS	Yuan+IS	MBI
1	45/47	34/36	51/53	39/41	51/53	46/48
10	137/145	70/75	93/99	89/96	78/81	71/75
100	F	122/126	F	104/113	102/105	159/166

Problem 3 shows the MBI update dominating for the given starting point. As the scaling increases, however, the Yuan+IS update is slightly better overall. Note the excellent performance of DW for the scaling 10, which, together with the MBI performance, indicates the positive effects of weak inverse updating when solving this problem.

Problem 3. *Box* 3D *function* (*dimension* $= 3$). Starting point: $(0, 10, 20)$. Exact minimizer: several.

Scaling	BFGS	DW	Yuan	BFGS+IS	Yuan+IS	MBI
1	40/44	36/39	39/42	37/40	38/40	32/34
10	59/69	59/68	68/75	71/80	69/75	66/71
100	F	41/46	F	40/45	38/42	40/44

Like Problem 1, the next problem also illustrates the importance of a sizing strategy. Furthermore, it is an example where the sized *and* interpolating updates seem to outperform the others. In particular, for large scaling, the MBI update is second only to the Yuan+IS update.

Problem 4. *Penalty function* I (*dimension* $= 10$). Starting point: $(1, 2, \ldots, 10)$. Exact minimizer: not available.

Scaling	BFGS	DW	Yuan	BFGS+IS	Yuan+IS	MBI
1	65/74	61/74	58/66	62/74	61/70	60/69
10	F	71/73	F	70/72	71/74	68/71
100	F	242/246	F	276/278	170/172	206/208

Problem 5 illustrates the same advantages for the interpolating updates as Problem 4, but in this case the MBI update dominates for the large scaling of the initial guess, while being competitive for the other scalings.

Problem 5. *Trigonometric function* (*dimension* $= 10$). Starting point: $(1, \frac{1}{2}, \ldots, \frac{1}{10})$. Exact minimizer: not available.

Scaling	BFGS	DW	Yuan	BFGS+IS	Yuan+IS	MBI
1	23/25	26/28	23/25	27/29	26/28	25/28
10	47/51	40/43	90/93	40/43	41/44	42/45
100	145/149	103/107	143/145	104/108	56/59	52/54

Finally, Problem 6 is an example where the MBI update seems to be the overall best update.

Problem 6. *Brown and Dennis function* (*dimension* $= 4$). Starting point: $(25, 5, -5, -1)$. Exact minimizer: not available.

Scaling	BFGS	DW	Yuan	BFGS+IS	Yuan+IS	MBI
1	44/46	31/33	42/44	33/35	32/34	29/31
10	95/97	64/66	94/96	66/68	65/67	62/64
100	F	84/86	408/410	91/93	88/90	85/87

Overall, from these results, we observe that interpolation can improve the performance of a quasi-Newton algorithm using a Hessian update satisfying it.

Among the interpolating updates, we see that the MBI update consistently outperforms both the Yuan and Yuan+IS updates for the usual starting point of these problems, but is sometimes superseded by Yuan+IS for a large scaling of the initial guess.

Finally, we report the following data to exemplify the behaviour of t_k for the MBI update. This data comes from minimizing the Wood function (Problem 1) with the initial guess suggested in [5].

Iteration	t_k	Iteration	t_k
1	0.930058	19	1.00128
2	0.888828	20	0.993555
3	0.901655	21	0.999559
4	0.897605	22	1.00006
5	0.897991	23	1
6	0.894728	24	1
7	0.885345	25	1
8	0.84989	26	1
9	0.528055	27	1
10	1.90101	28	1.00002
11	0.717606	29	1.00012
12	1.18575	30	1.00038
13	1.0616	31	1.00044
14	0.996262	32	0.999271
15	0.995617	33	0.999504
16	1.03186	34	0.999988
17	1.00877	35	0.999965
18	1.0261	36	1

5. Conclusion. We have obtained a new family of Hessian updates satisfying the interpolation condition introduced by Yuan in [8] by generalizing his development to the well-known Broyden class. We then introduced the MBI update, a specific choice of update in this class that simultaneously incorporates the weak inverse updating techniques introduced by Dennis and Wolkowicz in [2]. Our numerical results showed the merits of these two approaches and illustrated the performance of the MBI update on typical test problems. We conclude that the MBI update seems to offer a slight improvement for certain problems over the other updates tested.

Acknowledgments. The author would like to thank his supervisor Henry Wolkowicz for introducing him to the subject of this paper and for his patience and support during our work together at the University of Waterloo. This paper was much improved thanks to the many comments and suggestions of both Chris Paige and Henry Wolkowicz and by the critiques and suggestions of the editor and two anonymous referees.

REFERENCES

[1] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[2] J. E. DENNIS, JR., AND H. WOLKOWICZ, *Sizing and least change secant methods*, CORR 90-02, C&O Research Report, University of Waterloo, Waterloo, Ontario, 1990.

[3] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, New York, 1981.

[4] D. G. LUENBERGER, *Linear and nonlinear programming*, 2nd ed., Addison-Wesley, Reading, MA, 1984.

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

[6] S. OREN AND D. LUENBERGER, *Self-scaling variable metric algorithms–Part* I: *Criteria and sufficient conditions for scaling a class of algorithms*, Management Sci., 20 (1974), pp. 845–862.

[7] R. B. SCHNABEL, *Conic methods for unconstrained minimization and tensor methods for nonlinear equations*, Mathematical Programming: The State of The Art, Springer-Verlag, Berlin, New York, 1983.

[8] Y.-X. YUAN, *A modified* BFGS *algorithm for unconstrained optimization*, IMA J. Numer. Anal., 11 (1991), pp. 325–332.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1368–1393, November 1993

FAST FOURIER TRANSFORMS FOR NONEQUISPACED DATA*

A. DUTT[†] AND V. ROKHLIN[†]

Abstract. A group of algorithms is presented generalizing the fast Fourier transform to the case of noninteger frequencies and nonequispaced nodes on the interval $[-\pi, \pi]$. The schemes of this paper are based on a combination of certain analytical considerations with the classical fast Fourier transform and generalize both the forward and backward FFTs. Each of the algorithms requires $O(N \cdot \log N + N \cdot \log(1/\varepsilon))$ arithmetic operations, where ε is the precision of computations and N is the number of nodes. The efficiency of the approach is illustrated by several numerical examples.

Key words. fast Fourier transform, Fourier analysis, trigonometric series, interpolation, approximation theory

AMS subject classifications. 65R10, 65T10, 65T20, 65Y20

1. Introduction.

Fourier techniques have been a popular analytical tool in the study of physics and engineering for more than two centuries. A reason for the usefulness of such techniques is that the trigonometric functions $e^{i\omega x}$ are eigenfunctions of the differentiation operator and can be effectively used to model solutions of differential equations which arise in the fields mentioned above.

More recently, the arrival of digital computers and the development of the fast Fourier transform (FFT) algorithm in the 1960s (see [6]) have established Fourier analysis as a powerful and practical numerical tool. The FFT, which computes discrete Fourier transforms (DFTs), is now central to many areas, most notably spectral analysis and signal processing. In some applications, however, the input data is not uniformly spaced, a condition that is required for the FFT. In this paper we present a set of algorithms for computing more efficiently some generalizations of the DFT, namely, the forward and inverse transformations described by the equations

$$f_j = \sum_{k=0}^{N} \alpha_k \cdot e^{i\omega_k x_j} \tag{1}$$

for $j = 0, \ldots, N$, where $f_j \in \mathbf{C}$, $\alpha_k \in \mathbf{C}$, $\omega_k \in [-N/2, N/2]$, and $x_j \in [-\pi, \pi]$. Each algorithm requires a number of arithmetic operations proportional to

$$N \cdot \log N + N \cdot \log\left(\frac{1}{\varepsilon}\right), \tag{2}$$

where ε is the desired accuracy, compared with $O(N^2)$ operations required for the direct evaluation of (1) and $O(N^3)$ for the direct inversion.

Remark 1.1. The DFT can be described by either of the two closely related formulae

$$f_j = \sum_{k=0}^{N-1} \alpha_k \cdot e^{2\pi ikj/N} \tag{3}$$

*Received by the editors June 17, 1992; accepted for publication (in revised form) November 19, 1992.

[†]Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, Connecticut 06520. The work of these authors was supported in part by Office of Naval Research grant N00014-89-J-1527 and in part by National Science Foundation grant DMS-9012751.

for $j = 0, \dots, N-1$, and

$$f_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{2\pi ikj/N} \tag{4}$$

for $j = -N/2, \dots, N/2-1$. While the form (3) is normally used when the FFT is discussed, the form (4) is usually preferred in applications of the DFT to analysis (see, for example, [2], [9]).

Remark 1.2. The FFT algorithm reduces the number of operations for the DFT from $O(N^2)$ to $O(N \cdot \log N)$ by a sequence of algebraic manipulations (for more details on FFTs, see [4], [6], [7], [12]–[14]). In the more general case of (1), the structure of the linear transformation is also exploitable, and the algorithms of this paper combine certain analytical results with the existing FFT.

The plan of the paper is as follows. We start in §2 with some results from the analysis and approximation theory which are used in the design of the algorithms. An exact statement of the problem in §3 is then followed by informal descriptions of the algorithms in §4. In §5 we introduce some notation that is used in a set of more detailed algorithm descriptions in §6. Six numerical examples are presented in §7 to illustrate the performance of the schemes. Finally, §8 lists some generalizations and conclusions.

2. Mathematical and numerical preliminaries.

2.1. Elementary analytical tools.

In this subsection we summarize some well-known results to be used in the remainder of the paper. Lemmas 2.1 and 2.2 are obvious, and Lemmas 2.3 and 2.4 can be found, for example, in [10].

LEMMA 2.1. *For any real c,*

$$\int_{-\pi}^{\pi} e^{icx} dx = \frac{2}{c} \sin(c\pi). \tag{5}$$

LEMMA 2.2. *For any integer k,*

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{ikx} dx = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

LEMMA 2.3. *For any real $b > 0$ and complex z,*

$$\int_{-\infty}^{\infty} e^{-bx^2} \cdot e^{zx} dx = \sqrt{\frac{\pi}{b}} \cdot e^{z^2/4b}. \tag{7}$$

LEMMA 2.4. *For any real $b > 0$ and $a > 0$,*

$$\int_{a}^{\infty} e^{-bx^2} dx < \frac{e^{-ba^2}}{2ba}. \tag{8}$$

Proof.

$$\int_{a}^{\infty} e^{-bx^2} dx = \int_{0}^{\infty} e^{-b(x+a)^2} dx < e^{-ba^2} \int_{0}^{\infty} e^{-2bax} dx = \frac{e^{-ba^2}}{2ba}. \tag{9}$$ □

2.2. Relevant facts from approximation theory. The principal tool of this paper is a somewhat detailed analysis of Fourier series of functions $\phi : [-\pi, \pi] \to \mathbf{C}$ given by the formula

$$\phi(x) = e^{-bx^2} \cdot e^{icx}, \tag{10}$$

where $b > \frac{1}{2}$ and c are real numbers. We present this analysis in the lemmas and theorems of this subsection, numbered 2.5–2.10.

Lemmas 2.5 and 2.6 provide two inequalities that are used in Theorem 2.7, and Theorems 2.7–2.9 are intermediate results leading to Theorem 2.10. This final theorem explains how to approximate functions of the form e^{icx} using a small number of terms, and the algorithms of this paper are based upon this result. We derive error bounds for all approximations that allow us to perform numerical computations to any specified accuracy.

LEMMA 2.5. *For any real $b > \frac{1}{2}$, c and any integer k,*

$$\left|2\int_{\pi}^{\infty} e^{-bx^2} \cos((c-k)x)dx + e^{-b\pi^2} \cdot \int_{-\pi}^{\pi} e^{i(c-k)x}dx\right| < 2\pi e^{-b\pi^2} \cdot \left(1 + \frac{1}{\pi^2}\right). \tag{11}$$

Proof. Using the triangle inequality and Lemma 2.4, we have

$$\begin{aligned} &\left|2\int_{\pi}^{\infty} e^{-bx^2} \cdot \cos((c-k)x)dx + e^{-b\pi^2} \cdot \int_{-\pi}^{\pi} e^{i(c-k)x}dx\right| \\ &\le 2\int_{\pi}^{\infty} e^{-bx^2}dx + 2\pi e^{-b\pi^2} < 2\pi e^{-b\pi^2} \cdot \left(\frac{1}{2b\pi^2} + 1\right) \\ &< 2\pi e^{-b\pi^2} \cdot \left(\frac{1}{\pi^2} + 1\right). \quad \square \end{aligned} \tag{12}$$

LEMMA 2.6. *For any real $b > \frac{1}{2}$, c and any integer k,*

(13)

$$\left|2\int_{\pi}^{\infty} e^{-bx^2} \cos((c-k)x)dx + e^{-b\pi^2} \cdot \int_{-\pi}^{\pi} e^{i(c-k)x}dx\right| < \frac{8b\pi e^{-b\pi^2}}{(c-k)^2} \cdot \left(1 + \frac{1}{\pi^2}\right).$$

Proof. Integrating by parts, we have

$$\begin{aligned} &2\int_{\pi}^{\infty} e^{-bx^2} \cdot \cos((c-k)x)dx \\ &= \frac{2}{c-k}\left[e^{-bx^2} \sin((c-k)x)\right]_{\pi}^{\infty} + \frac{4b}{c-k}\int_{\pi}^{\infty} xe^{-bx^2} \sin((c-k)x)dx \\ &= -\frac{2}{c-k}e^{-b\pi^2} \sin((c-k)\pi) + \frac{4b}{c-k}\int_{\pi}^{\infty} xe^{-bx^2} \sin((c-k)x)dx. \end{aligned} \tag{14}$$

After rearranging the terms in (14) and integrating by parts again, we obtain

(15)

$$\left|2\int_{\pi}^{\infty} e^{-bx^2} \cos((c-k)x)dx + \frac{2e^{-b\pi^2}}{c-k} \sin((c-k)\pi)\right|$$

$$= \left| \frac{4b}{c-k} \int_\pi^\infty xe^{-bx^2} \sin((c-k)x)dx \right|$$

$$= \left| -\frac{4b}{(c-k)^2} \left(\left[xe^{-bx^2} \cos((c-k)x) \right]_\pi^\infty - \int_\pi^\infty (1-2bx^2)e^{-bx^2} \cos((c-k)x)dx \right) \right|$$

$$\leq \frac{4b}{(c-k)^2} \left(\pi e^{-b\pi^2} + \int_\pi^\infty e^{-bx^2} dx + \int_\pi^\infty x \cdot 2bxe^{-bx^2} dx \right)$$

$$< \frac{4b}{(c-k)^2} \left(\pi e^{-b\pi^2} + \int_\pi^\infty e^{-bx^2} dx + \left[-xe^{-bx^2} \right]_\pi^\infty + \int_\pi^\infty e^{-bx^2} dx \right).$$

Finally, due to (15) and Lemmas 2.1 and 2.4, we have
(16)

$$\left| 2 \int_\pi^\infty e^{-bx^2} \cos((c-k)x)dx + e^{-b\pi^2} \cdot \int_{-\pi}^{\pi} e^{i(c-k)x} dx \right| < \frac{4be^{-b\pi^2}}{(c-k)^2} \cdot \left(2\pi + \frac{2}{2b\pi} \right)$$
$$< \frac{8b\pi e^{-b\pi^2}}{(c-k)^2} \cdot \left(1 + \frac{1}{\pi^2} \right). \quad \square$$

The following theorem provides an explicit expression for the coefficients of a Fourier series that approximates functions of the form (10).

THEOREM 2.7. *Let $\phi(x) = e^{-bx^2} e^{icx}$ for any real $b > \frac{1}{2}$, c. Then, for any $x \in (-\pi, \pi)$,*

$$\left| \phi(x) - \sum_{k=-\infty}^{\infty} \rho_k e^{ikx} \right| < e^{-b\pi^2} \cdot \left(4b + \frac{70}{9} \right), \tag{17}$$

where

$$\rho_k = \frac{1}{2\sqrt{b\pi}} e^{-(c-k)^2/4b} \tag{18}$$

for $k = -\infty, \ldots, \infty$.

Proof. We denote by σ_k the kth Fourier coefficient for ϕ, so that for $x \in (-\pi, \pi)$,

$$\phi(x) = \sum_{k=-\infty}^{\infty} \sigma_k e^{ikx}, \tag{19}$$

and due to Lemma 2.3 and (18), we have
(20)

$$\sigma_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} \phi(x) e^{-ikx} dx$$
$$= \frac{1}{2\pi} \left(\int_{-\infty}^{\infty} e^{-bx^2} e^{icx} e^{-ikx} dx - \int_{-\infty}^{-\pi} e^{-bx^2} e^{icx} e^{-ikx} dx - \int_\pi^\infty e^{-bx^2} e^{icx} e^{-ikx} dx \right)$$
$$= \frac{1}{2\pi} \left(\sqrt{\frac{\pi}{b}} \cdot e^{-(c-k)^2/4b} + \int_\infty^\pi e^{-bx^2-icx+ikx} dx - \int_\pi^\infty e^{-bx^2+icx-ikx} dx \right)$$
$$= \rho_k - \frac{1}{\pi} \int_\pi^\infty e^{-bx^2} \cos((c-k)x)dx.$$

Rearranging (20) and applying Lemmas 2.5 and 2.6, we obtain the inequalities

$$\left|\sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{icx}e^{-ikx}dx\right| \quad < \quad e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi^2}\right), \tag{21}$$

$$\left|\sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{icx}e^{-ikx}dx\right| \quad < \quad \frac{4be^{-b\pi^2}}{(c-k)^2}\cdot\left(1+\frac{1}{\pi^2}\right), \tag{22}$$

and it now follows from the combination of (19), (21), and (22) that, for any $x \in (-\pi,\pi)$,

$$\begin{aligned} &\left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - e^{-b\pi^2}\cdot e^{icx}\right| \\ &= \left|\sum_{k=-\infty}^{\infty} e^{ikx}\cdot\left(\sigma_k - \rho_k - \frac{e^{-b\pi^2}}{2\pi}\int_{-\pi}^{\pi} e^{icx}e^{-ikx}dx\right)\right| \\ &< \sum_{k,|c-k|\geq 3}\frac{4be^{-b\pi^2}}{(c-k)^2}\cdot\left(1+\frac{1}{\pi^2}\right) + \sum_{k,|c-k|<3} e^{-b\pi^2}\cdot\left(1+\frac{1}{\pi^2}\right) \\ &< 4be^{-b\pi^2}\cdot\frac{9}{8}\cdot 2\cdot\sum_{k=3}^{\infty}\frac{1}{k^2} + 6e^{-b\pi^2}\cdot\frac{10}{9}. \end{aligned} \tag{23}$$

Some elementary analysis yields

$$\sum_{k=3}^{\infty}\frac{1}{k^2} < \frac{1}{9} + \int_3^{\infty}\frac{dx}{x^2} = \frac{1}{9}+\frac{1}{3} = \frac{4}{9}, \tag{24}$$

and substituting (24) into (23), we have

$$\left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - e^{-b\pi^2}\cdot e^{icx}\right| \quad < \quad e^{-b\pi^2}\cdot\left(4b+\frac{60}{9}\right). \tag{25}$$

To complete the proof we make use of the triangle inequality and (25) to obtain

$$\begin{aligned} \left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx}\right| &\leq \left|\phi(x) - \sum_{k=-\infty}^{\infty}\rho_k e^{ikx} - e^{-b\pi^2}\cdot e^{icx}\right| + \left|e^{-b\pi^2}\cdot e^{icx}\right| \\ &< e^{-b\pi^2}\cdot\left(4b+\frac{70}{9}\right). \quad \square \end{aligned} \tag{26}$$

Remark 2.1. According to Theorem 2.7, functions of the form $e^{-bx^2}e^{icx}$ can be approximated by a Fourier series whose coefficients are given analytically, and the error of the approximation decreases exponentially as b increases.

Remark 2.2. The coefficients ρ_k as defined by (18) have a peak at $k = [c]$, the nearest integer to c, and decay exponentially as $k \to \pm\infty$. We keep only the $q+1$ largest coefficients, where the integer q is chosen such that

$$q \geq 4b\pi, \tag{27}$$

so as to satisfy the inequality

$$e^{-(q/2)^2/4b} \leq e^{-b\pi^2}. \tag{28}$$

The following theorem estimates the truncation error under the conditions of Remark 2.2 and thus provides a way of approximating functions of the form (10) by a $(q+1)$-term series.

THEOREM 2.8. *Let $\phi(x) = e^{-bx^2} e^{icx}$ for any real $b > \frac{1}{2}$, c, and let q be an even integer such that $q \geq 4b\pi$. Then, for any $x \in (-\pi, \pi)$,*

$$\left| \phi(x) - \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx} \right| < e^{-b\pi^2} \cdot (4b+9), \tag{29}$$

where $\{\rho_k\}$ are defined by (18).

Proof. For any $x \in (-\pi, \pi)$,

$$\begin{aligned} &\left| \phi(x) - \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx} \right| \\ &\leq \left| \phi(x) - \sum_{k=-\infty}^{\infty} \rho_k e^{ikx} \right| + \left| \sum_{k>[c]+q/2} \rho_k e^{ikx} \right| + \left| \sum_{k<[c]-q/2} \rho_k e^{ikx} \right| . \end{aligned} \tag{30}$$

Due to (18) and the triangle inequality, we have the inequalities

$$\left| \sum_{k>[c]+q/2} \rho_k e^{ikx} \right| \leq \sum_{k=[c]+q/2+1}^{\infty} \frac{e^{-(c-k)^2/4b}}{2\sqrt{b\pi}} < \sum_{k=q/2}^{\infty} \frac{e^{-k^2/4b}}{\sqrt{2\pi}}, \tag{31}$$

$$\left| \sum_{k<[c]-q/2} \rho_k e^{ikx} \right| \leq \sum_{k=-\infty}^{[c]-q/2-1} \frac{e^{-(c-k)^2/4b}}{2\sqrt{b\pi}} < \sum_{k=q/2}^{\infty} \frac{e^{-k^2/4b}}{\sqrt{2\pi}}. \tag{32}$$

Some elementary analysis and an application of Lemma 2.4 yields

$$\sum_{k=q/2}^{\infty} e^{-k^2/4b} < e^{-(q/2)^2/4b} + \int_{q/2}^{\infty} e^{-x^2/4b} dx < e^{-(q/2)^2/4b} \cdot \left(1 + \frac{4b}{2q/2}\right), \tag{33}$$

and it follows from the combination of (27), (28), and (33) that

$$\sum_{k=q/2}^{\infty} e^{-k^2/4b} < e^{-b\pi^2} \cdot \left(1 + \frac{1}{\pi}\right). \tag{34}$$

Substituting (34) into (31) and (32), we have

$$\left| \sum_{k>[c]+q/2} \rho_k e^{ikx} \right| + \left| \sum_{k<[c]-q/2} \rho_k e^{ikx} \right| < \frac{2e^{-b\pi^2}}{\sqrt{2\pi}} \cdot \left(1 + \frac{1}{\pi}\right) < e^{-b\pi^2} \cdot \frac{10}{9}, \tag{35}$$

and finally, substituting (26) and (35) into (30), we obtain

$$\left| \phi(x) - \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx} \right| < e^{-b\pi^2} \cdot \left(4b + \frac{70}{9} + \frac{10}{9}\right) < e^{-b\pi^2} \cdot (4b+9). \tag{36}$$ □

The following corollary describes a formula for approximating e^{icx} using a series of $q+1$ terms.

COROLLARY 2.9. *Suppose that $m \geq 2$ is an integer and that the conditions of Theorem 2.8 are satisfied. Then, multiplying both sides of* (29) *by e^{bx^2}, we obtain*

$$\left| e^{icx} - e^{bx^2} \cdot \sum_{k=[c]-q/2}^{[c]+q/2} \rho_k e^{ikx} \right| < e^{bx^2} \cdot e^{-b\pi^2} \cdot (4b+9) \tag{37}$$
$$< e^{b\pi^2/m^2} \cdot e^{-b\pi^2} \cdot (4b+9)$$

for any $x \in [-\frac{\pi}{m}, \frac{\pi}{m}]$.

Finally, Theorem 2.10 makes use of a simple linear scaling to generalize the inequality (37) from $[-\frac{\pi}{m}, \frac{\pi}{m}]$ to any interval $[-d, d]$. This is the principal result of the section.

THEOREM 2.10. *Let $b > \frac{1}{2}, c, d > 0$ be real numbers, and let $m \geq 2, q \geq 4b\pi$ be integers. Then, for any $x \in [-d, d]$,*

$$\left| e^{icx} - e^{b(x\pi/md)^2} \cdot \sum_{k=[cmd/\pi]-q/2}^{[cmd/\pi]+q/2} \rho_k e^{ikx\pi/md} \right| < e^{-b\pi^2(1-1/m^2)} \cdot (4b+9) \tag{38}$$

where $\{\rho_k\}$ are defined by (18).

Remark 2.3. The error bounds obtained in the above theorems are rather pessimistic. Numerical estimates for the actual errors can be found in the Appendix to this paper.

3. Exact statement of the problem.

In the remainder of this paper we will operate under the following assumptions:

1. $\omega = \{\omega_0, \ldots, \omega_N\}$ and $x = \{x_0, \ldots, x_N\}$ are finite sequences of real numbers.
2. $\omega_k \in [-N/2, N/2]$ for $k = 0, \ldots, N$.
3. $x_j \in [-\pi, \pi]$ for $j = 0, \ldots, N$.
4. $\alpha = \{\alpha_0, \ldots, \alpha_N\}$, $f = \{f_{-N/2}, \ldots, f_{N/2}\}$, $\beta = \{\beta_{-N/2}, \ldots, \beta_{N/2}\}$, $g = \{g_0, \ldots, g_N\}$, $\gamma = \{\gamma_0, \ldots, \gamma_N\}$ and $h = \{h_0, \ldots, h_N\}$ are finite sequences of complex numbers.

We will consider the problems of applying and inverting the Fourier matrix and its transpose, i.e., we are interested in the transformations $F, G : \mathbf{C}^{N+1} \to \mathbf{C}^{N+1}$ and their inverses defined by the formulae

$$f_j = F(\alpha)_j = \sum_{k=0}^{N} \alpha_k \cdot e^{i\omega_k \cdot 2\pi j/N} \tag{39}$$

for $j = -N/2, \ldots, N/2$, and

$$g_j = G(\beta)_j = \sum_{k=-N/2}^{N/2} \beta_k \cdot e^{ikx_j} \tag{40}$$

for $j = 0, \ldots, N$.

Remark 3.1. If $x_k = -\omega_k \cdot 2\pi/N$ for $k = 0, \ldots, N$, then (39) can be rewritten as

$$f_j = \sum_{k=0}^{N} \alpha_k \cdot e^{-ijx_k}, \tag{41}$$

or alternatively as $F = G^*$.

We will also consider the more general transformation $H : \mathbf{C}^{N+1} \rightarrow \mathbf{C}^{N+1}$ defined by the formula

$$h_j = H(\gamma)_j = \sum_{k=0}^{N} \gamma_k \cdot e^{i\omega_k x_j}. \tag{42}$$

More formally, we consider the following problems:

- Problem 1. Given α, find $f = F(\alpha)$.
- Problem 2. Given β, find $g = G(\beta)$.
- Problem 3. Given γ, find $h = H(\gamma)$.
- Problem 4. Given f, find $\alpha = F^{-1}(f)$.
- Problem 5. Given g, find $\beta = G^{-1}(g)$.

Remark 3.2. We wish to perform all calculations with a fixed relative accuracy $\varepsilon > 0$. In the case of Problem 1, for instance, we are looking for a vector $\tilde{f} = \{\tilde{f}_{-N/2}, \ldots, \tilde{f}_{N/2}\}$ such that

$$\frac{||\tilde{f} - f||}{||f||} \leq \varepsilon. \tag{43}$$

In this sense, all algorithms described in this paper are approximate ones.

4. Informal descriptions of the algorithms. In this section we give informal outlines of algorithms for Problems 1–5 of §3. More formal descriptions of these algorithms are presented in §6.

4.1. Algorithms 1, 2, and 3 for Problems 1, 2, and 3. The algorithms for these problems are based on the following principal observation.

Observation 4.1. According to Theorem 2.10, any function of the form e^{icx} can be accurately represented on any finite interval on the real line using a small number of terms of the form $e^{bx^2} \cdot e^{ikx}$, and this number of terms, q, is independent of the value of c.

The FFT algorithm applies the Fourier matrix to arbitrary complex vectors in $O(N \log N)$ operations when $\{\omega_k\}$ are integers and $\{x_j\}$ are equally spaced in $[-\pi, \pi]$. For the efficient application of the transformations described by (39), (40), and (42), we relate these more general cases to the equispaced case of the FFT. Observation 4.1 is used in two ways to achieve this:

- to approximate each $e^{i\omega_k x}$ in terms of a q-term Fourier series;
- to approximate the value of a Fourier series at each x_j in terms of values at the nearest q equispaced nodes.

This interpolation between equispaced and nonequispaced sets of points can thus be performed in $O(Nq)$ operations.

Observation 4.2. The overall complexity of each such algorithm that couples the FFT with the interpolation scheme will be $O(N \log N + Nq)$ operations.

4.2. Algorithms 4 and 5 for Problems 4 and 5. Here we are interested in applying the complex matrices A^{-1} and $(A^*)^{-1}$ to arbitrary complex vectors where the elements of A are defined by

$$A_{jk} = e^{ikx_j} \tag{44}$$

for $j = 0, \ldots, N$ and $k = -N/2, \ldots, N/2$.

We make use of the following two simple observations.

Observation 4.3. The matrix AA^* is Toeplitz, and furthermore, its $2N+1$ distinct elements can be computed in $O(N \log N + Nq)$ operations due to Observation 4.2.

Proof. It is obvious from (44) that

$$(AA^*)_{jl} = \sum_{k=0}^{N} e^{ijx_k} \cdot e^{-ilx_k} = \sum_{k=0}^{N} e^{i(j-l)x_k}, \tag{45}$$

which is a function only of $(j-l)$, and is of the same form as (41), the description for Problem 1. □

Observation 4.4. From elementary matrix identities we see that

$$A^{-1} \equiv A^*(AA^*)^{-1}, \tag{46}$$

$$(A^*)^{-1} \equiv (AA^*)^{-1}A. \tag{47}$$

The Toeplitz matrix AA^* can be applied to arbitrary vectors in $O(N \log N)$ operations using an FFT-based discrete convolution. $(AA^*)^{-1}$ can therefore be applied to a vector in $O(\kappa(A) \cdot N \log N)$ operations using the conjugate gradient method where $\kappa(A)$ is the condition number of A.

Observation 4.5. A^{-1} and $(A^*)^{-1}$ can be applied to arbitrary vectors using $O(\kappa(A) \cdot N \log N + Nq)$ operations due to Observations 4.2 and 4.4.

Remark 4.6. It is well known that the condition number of A is 1 if the points $\{x_j\}$ are equally spaced. While the condition number deteriorates as the distribution of points becomes more nonuniform, in many cases of practical interest the points will be fairly uniformly spaced, so the condition number will not be very large.

4.3. Algorithm 6 for a variant of Problem 5. The following lemma describes a way of computing the coefficients of an $(N/2+1)$-term Fourier series that is tabulated at $N+1$ points.

LEMMA 4.1. *Suppose that the $N+1$ function values $g_0, \dots, g_N$ are given by the formula*

$$g_j = \sum_{k=-N/4}^{N/4} \beta_k \cdot e^{ikx_j}, \tag{48}$$

and the vector $\xi = \{\xi_0, \dots, \xi_N\}$ is the unique solution of the linear system described by the equations

$$\sum_{j=0}^{N} \xi_j \cdot e^{ikx_j} = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases} \tag{49}$$

for $k = -N/2, \dots, N/2$. Then, for $k = -N/4, \dots, N/4$,

$$\beta_k = \sum_{j=0}^{N} \xi_j \cdot g_j \cdot e^{-ikx_j}. \tag{50}$$

Proof. Substituting for g_j from (48), we have for $k = -N/4, \dots, N/4$

$$\sum_{j=0}^{N} \xi_j \cdot g_j \cdot e^{-ikx_j} = \sum_{j=0}^{N} \xi_j \cdot e^{-ikx_j} \cdot \sum_{l=-N/4}^{N/4} \beta_l \cdot e^{ilx_j} \tag{51}$$

$$= \sum_{l=-N/4}^{N/4} \beta_l \cdot \sum_{j=0}^{N} \xi_j \cdot e^{i(l-k)x_j} \tag{52}$$

$$= \beta_k. \quad \square \tag{53}$$

Remark 4.7. According to (49) and Lemma 2.2,

$$\sum_{j=0}^{N} \xi_j \cdot e^{ikx_j} = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{ikx} dx \tag{54}$$

for $k = -N/2, \ldots, N/2$. Thus, the set of numbers $\{\xi_j\}$ can be considered as quadrature weights that integrate exactly all Nth order trigonometric polynomials at the nodes $\{x_j\}$.

Observation 4.8. Rewriting (49) in matrix notation, we see that

$$A^T \xi = (0, \ldots, 0, 1, 0, \ldots, 0)^T = A^* \xi, \tag{55}$$

where $A_{jk} = e^{ikx_j}$, so the vector ξ is real and can be computed using $O(\kappa(A) \cdot N \log N + Nq)$ operations due to Observation 4.5.

Observation 4.9. Equation (50) is of the same form as (41). Thus, provided the vector ξ is known, the vector β can be computed in $O(N \log N + Nq)$ operations due to Observation 4.2.

Remark 4.10. According to Observation 4.9, if a function described by an $(N/2+1)$-term Fourier series is tabulated at $N+1$ arbitrary nodes, the $(N/2+1)$ coefficients can be obtained in $O(N \log N + Nq)$ operations. Also, due to Observation 4.8, the precomputation of the numbers $\{\xi_j\}$ needed for this algorithm requires $O(\kappa(A) \cdot N \log N + Nq)$ operations.

5. Notation. In this section we introduce the notation to be used in the next section for the detailed algorithm descriptions.

For an integer $m \geq 2$ and a real number $b > 0$, we will define a real number $\varepsilon > 0$ by

$$\varepsilon = e^{-b\pi^2(1-1/m^2)} \cdot (4b+9), \tag{56}$$

and we will denote by q the smallest even natural number such that

$$q \geq 4b\pi. \tag{57}$$

For an integer m and a set of real numbers $\{\omega_k\}$, we will denote by μ_k the nearest integer to $m\omega_k$ for $k = 0, \ldots, N$, and by $\{P_{jk}\}$ a set of real numbers defined by the formula

$$P_{jk} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(m\omega_k-(\mu_k+j))^2/4b} \tag{58}$$

for $k = 0, \ldots, N$ and $j = -q/2, \ldots, q/2$.

Observation 5.1. Setting $d = \pi$ in Theorem 2.10, we see that

$$\left| e^{i\omega_k x} - e^{b(x/m)^2} \cdot \sum_{j=-q/2}^{q/2} P_{jk} \cdot e^{i(\mu_k+j)x/m} \right| < \varepsilon \tag{59}$$

for any $k = 0, \dots, N$ and any $x \in [-\pi, \pi]$, where ε is defined by (56).

For a given set of complex numbers $\{\alpha_k\}$, we will denote by $\{\tau_j\}$ the unique set of complex coefficients such that

$$\sum_{k=1}^{N} \alpha_k \cdot \sum_{j=-q/2}^{q/2} P_{jk} \cdot e^{i(\mu_k+j)x/m} = \sum_{j=-mN/2}^{mN/2-1} \tau_j e^{ijx/m}, \tag{60}$$

so that

$$\tau_l = \sum_{j,k,\mu_k+j=l} \alpha_k \cdot P_{jk}. \tag{61}$$

We will denote by $\{T_j\}$ a set of complex numbers defined by the formula

$$T_j = \sum_{k=-mN/2}^{mN/2-1} \tau_k \cdot e^{2\pi ikj/mN} \tag{62}$$

for $j = -mN/2, \dots, mN/2 - 1$.

Furthermore, we will denote by $\{\tilde{f}_j\}$ another set of complex numbers defined by the formula

$$\tilde{f}_j = e^{b(2\pi j/mN)^2} \cdot T_j \tag{63}$$

for $j = -N/2, \dots, N/2$.

Observation 5.2. Combining (59)–(63) with the triangle inequality, we see that

$$|f_j - \tilde{f}_j| < \varepsilon \cdot \sum_{k=0}^{N} |\alpha_k| \tag{64}$$

for $j = -N/2, \dots, N/2$, where $\{f_j = F(\alpha)_j\}$ are defined by (39).

For an integer m and a set of real numbers $\{x_j\}$, we will denote by ν_j the nearest integer to $x_j mN/2\pi$ for $j = 0, \dots, N$, and by $\{Q_{jk}\}$ a set of real numbers defined by the formula

$$Q_{jk} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(x_j mN/2\pi - (\nu_j + k))^2/4b} \tag{65}$$

for $j = 0, \dots, N$ and $k = -q/2, \dots, q/2$.

Observation 5.3. Setting $d = N/2$ in Theorem 2.10, we see that

$$\left| e^{ikx_j} - e^{b(2\pi k/mN)^2} \cdot \sum_{l=-q/2}^{q/2} Q_{jk} \cdot e^{i(\nu_j+l)2\pi k/mN} \right| < \varepsilon \tag{66}$$

for any $j = 0, \dots, N$ and any $k \in [-N/2, N/2]$, where ε is defined by (56).

For a given set of complex numbers $\{\beta_k\}$, we will denote by $\{u_k\}$ a set of complex numbers defined by the formula

$$u_k = \beta_k \cdot e^{b(2\pi k/mN)^2} \tag{67}$$

for $k = -N/2, \dots, N/2$, and by $\{U_l\}$ a set of complex numbers defined by the formula

$$U_l = \sum_{k=-N/2}^{N/2} u_k \cdot e^{2\pi ikl/mN} \tag{68}$$

for $l = -mN/2, \dots, mN/2 - 1$.

Furthermore, we will denote by $\{\tilde{g}_j\}$ another set of complex numbers defined by the formula

$$\tilde{g}_j = \sum_{l=-q/2}^{q/2} Q_{jl} \cdot U_{\nu_j + l} \tag{69}$$

for $j = 0, \dots, N$.

Observation 5.4. Combining (66)–(69) with the triangle inequality, we see that

$$|g_j - \tilde{g}_j| < \varepsilon \cdot \sum_{k=0}^{N} |\beta_k| \tag{70}$$

for $j = 0, \dots, N$, where $\{g_j = G(\beta)_j\}$ are defined by (40).

For a set of real numbers $\{x_j\}$, we will denote by η_j the nearest integer to $x_j N/2\pi$ for $j = 0, \dots, N$, and by $\{R_{jk}\}$ a set of real numbers defined by the formula

$$R_{jk} = \frac{1}{2\sqrt{b\pi}} \cdot e^{-(x_j N/2\pi - (\eta_j + k))^2/4b} \tag{71}$$

for $j = 0, \dots, N$ and $k = -q/2, \dots, q/2$.

Observation 5.5. Setting $d = N/2$ in Theorem 2.10, we see that

$$\left| e^{ikx_j/m} - e^{b(2\pi k/mN)^2} \cdot \sum_{l=-q/2}^{q/2} R_{jk} \cdot e^{i(\eta_j + l)2\pi k/mN} \right| < \varepsilon \tag{72}$$

for any $j = 0, \dots, N$ and any $k \in [-N/2, N/2]$, where ε is defined by (56).

For a given set of complex numbers $\{\gamma_k\}$, we will denote by $\{v_j\}$ the unique set of complex coefficients such that

$$\sum_{k=0}^{N} \gamma_k \cdot \sum_{j=-q/2}^{q/2} P_{jk} \cdot e^{i(\mu_k + j)x/m} = \sum_{j=-mN/2}^{mN/2} v_j \cdot e^{ijx/m}, \tag{73}$$

so that

$$v_l = \sum_{j,k,\eta_k + j = l} \gamma_k \cdot P_{jk}. \tag{74}$$

We denote by $\{V_l\}$ a set of complex numbers defined by the formula

$$V_l = \sum_{k=-mN/2}^{mN/2} v_k \cdot e^{b(2\pi k/m^2 N)^2} \cdot e^{2\pi ikl/m^2 N} \tag{75}$$

for $l = -m^2 N/2, \dots, m^2 N/2 - 1$.

Furthermore, we will denote by $\{\tilde{h}_j\}$ another set of complex numbers defined by the formula

$$\tilde{h}_j = e^{b(x_j/m)^2} \cdot \sum_{l=-q/2}^{q/2} R_{jl} \cdot V_{\eta_j+l} \tag{76}$$

for $j = 0, \dots, N$.

Observation 5.6. Combining (59) and (72)–(76) with the triangle inequality, we see that

$$|h_j - \tilde{h}_j| < \delta \cdot \sum_{k=0}^{N} |\gamma_k| \tag{77}$$

for $j = 0, \dots, N$, where $\{h_j = H(\gamma)_j\}$ are defined by (42), and

$$\delta = 2e^{-b\pi^2(1-2/m^2)} \cdot (4b + 9). \tag{78}$$

For a set of real numbers $\{x_j\}$, A will denote a complex matrix whose elements are given by

$$A_{jk} = e^{ikx_j} \tag{79}$$

for $k = -N/2, \dots, N/2$ and $j = 0, \dots, N$, and $a = \{a_{-N}, \dots, a_N\}$ will denote a set of complex numbers defined by the formula

$$a_k = \sum_{j=0}^{N} e^{ikx_j}. \tag{80}$$

Finally, $\xi = \{\xi_0, \dots, \xi_N\}$ will denote a real vector defined by

$$\xi = (A^*)^{-1}(0, \dots, 0, 1, 0, \dots, 0)^T. \tag{81}$$

Remark 5.7. It is clear from Observation 4.3 that

$$(AA^*)_{jk} = a_{j-k}. \tag{82}$$

6. Detailed descriptions of the algorithms.

This section contains step-by-step descriptions and operation counts for the six algorithms of this paper. In the tables below we will make use of the facts that $q \sim \log(\frac{1}{\varepsilon})$ and $m^2 \ll N$.

ALGORITHM 1.

Step	Complexity	Description
Init	$O(Nq)$	**Comment** [Input parameter is the vector $\{\omega_0, \dots, \omega_N\}$.]

```
Comment [Input parameter is the vector $\{\omega_0, \dots, \omega_N\}$.]
Choose precision $\varepsilon$ to be achieved.
Set $b \approx \log(1/\varepsilon)$ and $q = \lceil 4b\pi \rceil$.
do $k = 0, N$
    Determine $\mu_k$, the nearest integer to $m\omega_k$
    do $j = -q/2, q/2$
        Compute $P_{jk}$ according to (58)
    end do
end do
do $j = -N/2, N/2$
    Compute $e^{b(2\pi j/mN)^2}$
end do
```

```
1      $O(Nq)$             Comment [Input parameter is the vector $\{\alpha_0, \dots, \alpha_N\}$.]
                           Comment [Compute Fourier coefficients $\tau_j$.]
                           do $k = 0, N$
                              do $j = -q/2, q/2$
                                 $\tau_{\mu_k+j} \leftarrow \tau_{\mu_k+j} + P_{jk} \cdot \alpha_k$
                              end do
                           end do

2      $O(mN \log N)$      Comment [Evaluate this Fourier Series at equispaced points in $[-m\pi, m\pi]$ us-
                           ing inverse FFT of size $mN$.]
                           Compute $T_j = \sum_{k=-mN/2}^{mN/2-1} \tau_k \cdot e^{2\pi ikj/mN}$ for
                              $j = -mN/2, \dots, mN/2 - 1$.

3      $O(N)$              Comment [Scale the values at those points which lie in $[-\pi, \pi]$.]
                           do $j = -N/2, N/2$
                              $\tilde{f}_j = T_j \cdot e^{b(2\pi j/mN)^2}$
                           end do

Total  $O(N \cdot \log(\frac{1}{\varepsilon}) + mN \cdot \log N)$
```

ALGORITHM 2.

```
Step   Complexity          Description

Init   $O(Nq)$             Comment [Input parameter is the vector $\{x_0, \dots, x_N\}$.]
                           Choose precision $\varepsilon$ to be achieved.
                           Set $b \approx \log(1/\varepsilon)$ and $q = \lceil 4b\pi \rceil$.
                           do $j = 0, N$
                              Determine $\nu_j$, the nearest integer to $x_j mN/2\pi$
                              do $k = -q/2, q/2$
                                 Compute $Q_{jk}$ according to (65)
                              end do
                           end do
                           do $k = -N/2, N/2$
                              Compute $e^{b(2\pi k/mN)^2}$
                           end do

1      $O(N)$              Comment [Input parameter is the complex vector $\{\beta_{-N/2}, \dots, \beta_{N/2}\}$.]
                           Comment [Compute new, scaled Fourier coefficients.]
                           do $k = -N/2, N/2$
                              $u_k = \beta_k \cdot e^{b(2\pi k/mN)^2}$
                           end do

2      $O(mN \log N)$      Comment [Evaluate this Fourier Series at equispaced points in $[-\pi, \pi]$ using
                           inverse FFT of size $mN$.]
                           Compute $U_j = \sum_{k=-N/2}^{N/2} u_k \cdot e^{2\pi ikj/mN}$ for $j = -mN/2, \dots, mN/2 - 1$.

3      $O(Nq)$             Comment [Compute approximate values at desired points in terms of the values
                           at equispaced points.]
                           do $j = 0, N$
                              do $k = -q/2, q/2$
                                 $\tilde{g}_j \leftarrow \tilde{g}_j + Q_{jk} \cdot U_{\nu_j+k}$
                              end do
                           end do

Total  $O(mN \cdot \log N + N \cdot \log(\frac{1}{\varepsilon}))$
```

ALGORITHM 3.

```
Step  Complexity             Description
Init  $O(Nq)$                Comment [Input parameters are the vectors $\{\omega_0, \dots, \omega_N\}$
                             and $\{x_0, \dots, x_N\}$.]
                             Choose precision $\varepsilon$ to be achieved.
                             Set $b \approx \log(1/\varepsilon)$ and $q = \lceil 4b\pi \rceil$.
                             do $k = 0, N$
                                 Determine $\mu_k$, the nearest integer to $m\omega_k$
                                 do $j = -q/2, q/2$
                                     Compute $P_{jk}$ according to (58)
                                 end do
                             end do
                             do $k = -mN/2, mN/2$
                                 Compute $e^{b(2\pi k/m^2 N)^2}$
                             end do
                             do $j = 0, N$
                                 Determine $\eta_j$, the nearest integer to $x_j N/2\pi$
                                 do $k = -q/2, q/2$
                                     Compute $R_{jk}$ according to (71)
                                 end do
                             end do
                             do $j = 0, N$
                                 Compute $e^{b(x_j/m)^2}$
                             end do

1     $O(Nq)$                Comment [Input parameter is the vector $\{\gamma_0, \dots, \gamma_N\}$.]
                             Comment [Compute Fourier coefficients $v_j$.]
                             do $k = 0, N$
                                 do $j = -q/2, q/2$
                                     $v_{\mu_k+j} \leftarrow v_{\mu_k+j} + P_{jk} \cdot \gamma_k$
                                 end do
                             end do

2     $O(mN)$                Comment [Scale the coefficients.]
                             do $k = -mN/2, mN/2$
                                 $v_k \leftarrow v_k \cdot e^{b(2\pi k/m^2 N)^2}$
                             end do

3     $O(m^2 N \log N)$      Comment [Evaluate this Fourier Series at equispaced points in
                             $[-m\pi, m\pi]$ using inverse FFT of size $m^2 N$.]
                             Compute $V_j = \sum_{k=-mN/2}^{mN/2} v_k \cdot e^{2\pi i k j/m^2 N}$
                                 for $j = -m^2 N/2, \dots, m^2 N/2 - 1$.

4     $O(Nq)$                Comment [Compute approximate values at desired points in
                             terms of the values at equispaced points.]
                             do $j = 0, N$
                                 do $k = -q/2, q/2$
                                     $\tilde{h}_j \leftarrow \tilde{h}_j + R_{jk} \cdot V_{\eta_j+k}$
                                 end do
                             end do
```

5	$O(N)$	**Comment** [Scale the values.]

```
do $j = 0, N$
    $\tilde{h}_j \leftarrow \tilde{h}_j \cdot e^{b(x_j/m)^2}$
end do
```

Total $O(m^2N \cdot \log N + N \cdot \log(\frac{1}{\varepsilon}))$

ALGORITHM 4.

Step	Complexity	Description
Init	$O(N \log N + Nq)$	Initialization for Algorithm 1. Initialization for Algorithm 2. Compute elements $\{a_k\}$ of Toeplitz matrix $(AA^*)^{-1}$ as defined by (80) using Algorithm 1.
1	$O(N \log N + Nq)$	Compute Af using Algorithm 2.
2	$O(\kappa(A) \cdot N \log N)$	Compute $\tilde{\alpha} = (AA^*)^{-1}(Af)$ using Conjugate Gradient algorithm.
Total	$O(\kappa(A) \cdot N \cdot \log N + N \cdot \log(\frac{1}{\varepsilon}))$	

ALGORITHM 5.

Step	Complexity	Description
Init	$O(N \log N + Nq)$	Initialization for Algorithm 1. Compute elements $\{a_k\}$ of Toeplitz matrix $(AA^*)^{-1}$ as defined by (80) using Algorithm 1.
1	$O(\kappa(A) \cdot N \log N)$	Compute $(AA^*)^{-1}g$ using Conjugate Gradient algorithm.
2	$O(N \log N + Nq)$	Compute $\tilde{\beta} = A^*((AA^*)^{-1}g)$ using Algorithm 1.
Total	$O(\kappa(A) \cdot N \cdot \log N + N \cdot \log(\frac{1}{\varepsilon}))$	

ALGORITHM 6.

Step	Complexity	Description
Init	$O(\kappa(A) \cdot N \log N + Nq)$	Initialization for Algorithm 5. Compute ξ as defined by (81) using Algorithm 5.
1	$O(N)$	Compute $\hat{g}_j = \xi_j g_j$ for $j = 0, \ldots, N$.
2	$O(N \log N + Nq)$	Compute $\tilde{\beta} = A^* \hat{g}$ using Algorithm 1.
Total	$O(N \cdot \log N + N \cdot \log(\frac{1}{\varepsilon}))$	

The storage requirements of an algorithm are also an important characteristic. From the above descriptions for the initialization steps, the asymptotic storage requirements for each algorithm are of the form

$$\lambda \cdot N \cdot q, \tag{83}$$

where the coefficient λ is software- and hardware-dependent.

7. Implementation and numerical results. We have written FORTRAN implementations of the six algorithms of this paper in both single and double precision arithmetic and have applied these programs to a variety of situations. In this section we discuss several details of our implementations and demonstrate the performance of the algorithms with six numerical examples.

Several technical details of our implementations appear to be worth mentioning here.

1. Each implementation consists of two main subroutines: the first is an initialization stage in which the matrix operators of the algorithm are precomputed and stored, and the second is an evaluation stage in which these operators are applied. Successive application of the linear transformations to multiple vectors requires the initialization to be performed only once.

2. For the single precision versions of each algorithm our choice of parameters was $m = 2$, $b = 0.5993$, and $q = 10$. For double precision we chose $m = 2$, $b = 1.5629$, and $q = 28$.

3. The algorithms as described in this paper all require an FFT of size proportional to N, and thus will perform efficiently whenever the FFT does. This restriction on the FFT size can be removed by extending the input vector to length $2^{\lceil \log_2 N \rceil}$ (i.e., the smallest power of 2 which is greater than N) and padding it with zeroes. This ensures that the algorithms will perform efficiently for any choice of N. In our implementations, these changes were made.

4. Each of the algorithms of this paper requires the evaluation of sums of the form

$$f_j = \sum_{k=-N/2}^{N/2-1} \alpha_k \cdot e^{2\pi ikj/N} \tag{84}$$

for $j = -N/2, \dots, N/2-1$, whereas most FFT software computes sums of the form

$$f_j = \sum_{k=0}^{N-1} \alpha_k \cdot e^{2\pi ikj/N} \tag{85}$$

for $j = 0, \dots, N-1$. We used a standard FFT to evaluate sums of the form (84) by defining $\hat{\alpha}_k = \alpha_k$ for $k = 0, \dots, N/2-1$, $\hat{\alpha}_k = \alpha_{k-N}$ for $k = N/2, \dots, N-1$, $\hat{f}_j = f_j$ for $j = 0, \dots, N/2-1$, and $\hat{f}_j = f_{k-N}$ for $j = N/2, \dots, N-1$. This substitution converts the form (84) to the form (85).

Our implementations of the algorithms of this paper have been tested on the Sun SPARCstation 1 for a variety of input data. Six experiments are described below and their results are summarized in Tables 1–6. These tables contain accuracies and CPU timings for the algorithms with computations performed in both single and double precision arithmetic, and the input size N varying between 64 and 4096. In addition, each table contains the CPU times required to solve the same set of problems via a direct calculation, and Tables 1–3 include timings for an FFT of the same size. Tables 1–3 also contain the accuracies of the direct single precision calculations.

Two measures of accuracy were chosen for each example. In Examples 1, 2, and 3, these are defined by the formulae

$$E_\infty = \max_{0 \le j \le N} |\tilde{f}_j - f_j| \Bigg/ \sum_{j=0}^{N} |\alpha_j| , \tag{86}$$

and

$$E_2 = \sqrt{\sum_{j=0}^{N} |\tilde{f}_j - f_j|^2 \Bigg/ \sum_{j=0}^{N} |f_j|^2}, \tag{87}$$

where α is the input vector, f is the result of a direct computation in double precision arithmetic, and $\tilde{f}$ is the result of the computation being considered.

In Examples 4, 5, and 6, they are defined by

$$E_\infty = \max_{0\le j\le N} |\tilde{\alpha}_j - \alpha_j| \Big/ \max_{0\le j\le N} |\alpha_j| \tag{88}$$

and

$$E_2 = \sqrt{\sum_{j=0}^{N} |\tilde{\alpha}_j - \alpha_j|^2 \Big/ \sum_{j=0}^{N} |\alpha_j|^2}, \tag{89}$$

where α is the input for a direct double precision computation, and $\tilde{\alpha}$ is the result of applying the algorithm to the result of this computation.

Remark 7.1. The formulae (86)–(89) measure fairly accurately the errors of all single precision computations. However, they can only provide rough estimates of the errors produced by the double precision versions of the algorithms.

Remark 7.2. In the direct methods for Problems 1 and 2, we used the fact that $e^{ikx_j} = (e^{ix_j})^k$ to reduce the number of exponential computations from N^2 to N. N^2 exponentials are required in the direct method for Problem 3, and, for larger N, the available memory on the machine is insufficient for the precomputation and storage of these numbers. The direct implementation we used for this problem computes each exponential when it is needed.

Remark 7.3. Standard LINPACK Gaussian elimination subroutines were used as the direct methods for comparing timings in Examples 4, 5 and 6. Estimated timings are presented for larger N, where this computation became impractical.

Following are the descriptions of the experiments and the tables of numerical results.

Example 1. Here we consider the transformation $F : C^{N+1} \to C^{N+1}$ of Problem 1 as defined by the formula

$$F(\alpha)_j = \sum_{k=0}^{N} \alpha_k \cdot e^{i\omega_k \cdot 2\pi j/N} \tag{90}$$

for $j = -N/2, \ldots, N/2$. In this example, $\{\omega_0, \ldots, \omega_N\}$ were randomly distributed on the interval $[-N/2, N/2]$, and $\{\alpha_0, \ldots, \alpha_N\}$ were generated randomly on the unit square in the complex plane defined by the formulae

$$0 \le \mathrm{Re}(z) \le 1, \qquad 0 \le \mathrm{Im}(z) \le 1. \tag{91}$$

The results of applying Algorithm 1 to this problem are presented in Tables 1(a) and 1(b).

Example 2. Here we consider the transformation $G : \mathbf{C}^{N+1} \to \mathbf{C}^{N+1}$ of Problem 2 as defined by the formula

$$G(\beta)_j = \sum_{k=-N/2}^{N/2} \beta_k \cdot e^{ikx_j} \tag{92}$$

for $j = 0, \ldots, N$. In this example, $\{x_0, \ldots, x_N\}$ were randomly distributed on the interval $[-\pi, \pi]$, and $\{\beta_{-N/2}, \ldots, \beta_{N/2}\}$ were generated randomly on the unit square in the complex plane defined by the formulae

$$0 \le \mathrm{Re}(z) \le 1, \qquad 0 \le \mathrm{Im}(z) \le 1. \tag{93}$$

TABLE 1(a)
Example 1. *Single precision computations.*

N	Errors				Timings (sec.)			
	Algorithm		Direct		Algorithm		Direct	FFT
	E_∞	E_2	E_∞	E_2	Init.	Eval.		
64	0.959 E−06	0.149 E−05	0.411 E−06	0.979 E−06	0.011	0.003	0.01	0.0008
128	0.108 E−05	0.265 E−05	0.724 E−06	0.182 E−05	0.022	0.006	0.03	0.0015
256	0.122 E−05	0.329 E−05	0.996 E−06	0.370 E−05	0.044	0.014	0.13	0.0034
512	0.176 E−05	0.796 E−05	0.154 E−05	0.691 E−05	0.088	0.029	0.49	0.0078
1024	0.199 E−05	0.113 E−04	0.197 E−05	0.146 E−04	0.174	0.065	1.90	0.0174
2048	0.255 E−05	0.230 E−04	0.317 E−05	0.280 E−04	0.348	0.136	7.47	0.0352
4096	0.427 E−05	0.431 E−04	0.506 E−05	0.555 E−04	0.701	0.315	30.24	0.0846

TABLE 1(b)
Example 1. *Double precision computations.*

N	Errors		Timings (sec.)			
	E_∞	E_2	Alg. init.	Alg. eval.	Direct	FFT
64	0.602 E−14	0.638 E−14	0.036	0.008	0.02	0.001
128	0.356 E−14	0.715 E−14	0.075	0.016	0.06	0.002
256	0.437 E−14	0.946 E−14	0.148	0.034	0.22	0.005
512	0.519 E−14	0.160 E−13	0.297	0.075	0.84	0.012
1024	0.518 E−14	0.314 E−13	0.600	0.155	3.23	0.026
2048	0.755 E−14	0.631 E−13	1.204	0.322	12.55	0.059
4096	0.118 E−13	0.125 E−12	2.418	0.713	49.69	0.132

TABLE 2(a)
Example 2. *Single precision computations.*

N	Errors				Timings (sec.)			
	Algorithm		Direct		Algorithm		Direct	FFT
	E_∞	E_2	E_∞	E_2	Init.	Eval.		
64	0.870 E−06	0.176 E−05	0.147 E−06	0.298 E−06	0.011	0.003	0.01	0.0008
128	0.148 E−05	0.199 E−05	0.541 E−06	0.764 E−06	0.022	0.005	0.03	0.0015
256	0.780 E−06	0.349 E−05	0.414 E−06	0.114 E−05	0.043	0.012	0.12	0.0034
512	0.953 E−06	0.890 E−05	0.828 E−06	0.334 E−05	0.086	0.027	0.46	0.0078
1024	0.182 E−05	0.103 E−04	0.311 E−05	0.534 E−05	0.174	0.055	1.80	0.0174
2048	0.209 E−05	0.181 E−04	0.685 E−05	0.923 E−05	0.345	0.124	7.11	0.0352
4096	0.427 E−05	0.318 E−04	0.211 E−04	0.222 E−04	0.687	0.283	29.03	0.0846

TABLE 2(b)
Example 2. *Double precision compuations.*

N	Errors		Timings (sec.)			
	E_∞	E_2	Alg. init.	Alg. eval.	Direct	FFT
64	0.249 E−14	0.814 E−14	0.038	0.005	0.01	0.001
128	0.501 E−14	0.746 E−14	0.075	0.012	0.05	0.002
256	0.418 E−14	0.623 E−14	0.148	0.028	0.18	0.005
512	0.356 E−14	0.831 E−14	0.297	0.060	0.69	0.012
1024	0.793 E−14	0.192 E−13	0.596	0.126	2.72	0.026
2048	0.138 E−13	0.405 E−13	1.188	0.264	10.86	0.059
4096	0.278 E−13	0.904 E−13	2.387	0.573	43.36	0.132

The results of applying Algorithm 2 to this problem are presented in Tables 2(a) and 2(b).

Example 3. Here we consider the transformation $H : \mathbf{C}^{N+1} \to \mathbf{C}^{N+1}$ of Problem 3 as defined by the formula

$$H(\gamma)_j = \sum_{k=0}^{N} \gamma_k \cdot e^{i\omega_k x_j} \tag{94}$$

for $j = 0, \ldots, N$. In this example, $\{\omega_0, \ldots, \omega_N\}$ were randomly distributed on the interval $[-N/2, N/2]$, $\{x_0, \ldots, x_N\}$ were randomly distributed on the interval $[-\pi, \pi]$, and $\{\gamma_0, \ldots, \gamma_N\}$ were generated randomly on the unit square in the complex plane defined by the formulae

$$0 \le \mathrm{Re}(z) \le 1, \qquad 0 \le \mathrm{Im}(z) \le 1. \tag{95}$$

The results of applying Algorithm 3 to this problem are presented in Tables 3(a) and 3(b).

TABLE 3(a)
Example 3. *Single precision computations.*

N	Errors				Timings (sec.)			
	Algorithm		Direct		Algorithm		Direct	FFT
	E_∞	E_2	E_∞	E_2	Init.	Eval.		
64	0.133 E−05	0.232 E−05	0.341 E−06	0.673 E−06	0.022	0.007	0.12	0.0008
128	0.154 E−05	0.362 E−05	0.405 E−06	0.121 E−05	0.044	0.013	0.46	0.0015
256	0.152 E−05	0.550 E−05	0.736 E−06	0.292 E−05	0.087	0.029	1.90	0.0034
512	0.188 E−05	0.956 E−05	0.113 E−05	0.642 E−05	0.177	0.061	7.79	0.0078
1024	0.277 E−05	0.151 E−04	0.163 E−05	0.110 E−04	0.352	0.131	32.04	0.0174
2048	0.734 E−05	0.364 E−04	0.290 E−05	0.267 E−04	0.706	0.299	131.41	0.0352
4096	0.828 E−05	0.726 E−04	0.461 E−05	0.526 E−04	1.404	0.644	532.95	0.0846

TABLE 3(b)
Example 3. *Double precision computations.*

N	Errors		Timings (sec.)			
	E_∞	E_2	Alg. init.	Alg. eval.	Direct	FFT
64	0.166 E−13	0.226 E−13	0.074	0.015	0.20	0.001
128	0.252 E−13	0.216 E−13	0.153	0.034	0.79	0.002
256	0.318 E−13	0.315 E−13	0.302	0.069	3.18	0.005
512	0.131 E−13	0.289 E−13	0.601	0.146	12.76	0.012
1024	0.203 E−13	0.425 E−13	1.210	0.297	51.12	0.026
2048	0.324 E−13	0.801 E−13	2.403	0.643	205.17	0.059
4096	0.244 E−13	0.124 E−12	4.824	1.326	827.52	0.132

Example 4. Here we consider Problem 4 of §3. In this example, the numbers $\{\omega_k\}$ were defined by the formula

$$\omega_k = -\frac{N}{2} + (k + 0.5 + \delta_k) \cdot \frac{N}{N+1} \tag{96}$$

for $k = 0, \ldots, N$, where δ_k were randomly distributed on the interval $[-0.1, 0.1]$. In addition, the numbers $\{\alpha_0, \ldots, \alpha_N\}$ were generated randomly on the unit square in the complex plane defined by the formulae

$$0 \le \mathrm{Re}(z) \le 1, \qquad 0 \le \mathrm{Im}(z) \le 1, \tag{97}$$

and the numbers $\{f_{-N/2}, \ldots, f_{N/2}\}$ were computed directly in double precision arithmetic according to the formula

$$f_j = \sum_{k=0}^{N} \alpha_k \cdot e^{i\omega_k \cdot 2\pi j/N}. \tag{98}$$

The vector f was then used as input for Algorithm 4. Results of this experiment are presented in Tables 4(a) and 4(b).

TABLE 4(a)
Example 4. Single precision computations.

N	Errors		Timings (sec.)		
	E_∞	E_2	Alg. init.	Alg. eval.	Direct
64	0.492 E−05	0.339 E−05	0.02	0.05	0.36
128	0.154 E−04	0.568 E−05	0.04	0.11	2.78
256	0.424 E−04	0.128 E−04	0.08	0.20	23.0
512	0.809 E−04	0.252 E−04	0.18	0.45	184
1024	0.203 E−03	0.474 E−04	0.36	0.82	1472 (est.)
2048	0.428 E−03	0.979 E−04	0.78	1.91	11776 (est.)
4096	0.106 E−02	0.195 E−03	1.66	4.64	94208 (est.)

TABLE 4(b)
Example 4. Double precision computations.

N	Errors		Timings (sec.)		
	E_∞	E_2	Alg. init.	Alg. eval.	Direct
64	0.143 E−13	0.109 E−13	0.07	0.17	0.37
128	0.208 E−13	0.149 E−13	0.11	0.34	2.96
256	0.493 E−13	0.256 E−13	0.20	0.75	23.6
512	0.121 E−12	0.500 E−13	0.48	1.67	189
1024	0.279 E−12	0.926 E−13	0.94	3.55	1512 (est.)
2048	0.593 E−12	0.192 E−12	1.95	7.75	12096 (est.)
4096	0.138 E−11	0.375 E−12	4.02	18.28	96768 (est.)

Example 5. Here we consider Problem 5 of §3. In this example, the numbers $\{x_j\}$ were defined by the formula

$$x_j = -\pi + 2\pi \cdot \frac{j + 0.5 + \delta_j}{N+1} \tag{99}$$

for $j = 0, \ldots, N$, where δ_j were randomly distributed on the interval $[-0.1, 0.1]$. In addition, the numbers $\{\beta_{-N/2}, \ldots, \beta_{N/2}\}$ were generated randomly on the unit square in the complex plane defined by the formulae

$$0 \le \mathrm{Re}(z) \le 1, \qquad 0 \le \mathrm{Im}(z) \le 1, \tag{100}$$

and the numbers $\{g_0, \ldots, g_N\}$ were computed directly in double precision arithmetic according to the formula

$$g_j = \sum_{k=-N/2}^{N/2} \beta_k \cdot e^{ikx_j}. \tag{101}$$

The vector g was then used as input for Algorithm 5. Results of this experiment are presented in Tables 5(a) and 5(b).

TABLE 5(a)
Example 5. *Single precision computations.*

N	Errors		Timings (sec.)		
	E_∞	E_2	Alg. init.	Alg. eval.	Direct
64	0.212 E−05	0.183 E−05	0.02	0.06	0.36
128	0.117 E−04	0.523 E−05	0.04	0.10	2.78
256	0.190 E−04	0.955 E−05	0.09	0.19	23.0
512	0.287 E−04	0.202 E−04	0.19	0.41	184
1024	0.560 E−04	0.376 E−04	0.37	0.83	1472 (est.)
2048	0.106 E−03	0.752 E−04	0.78	1.90	11776 (est.)
4096	0.225 E−03	0.148 E−03	1.66	4.67	94208 (est.)

TABLE 5(b)
Example 5. *Double precision computations.*

N	Errors		Timings (sec.)		
	E_∞	E_2	Alg. init.	Alg. eval.	Direct
64	0.310 E−13	0.120 E−13	0.06	0.18	0.37
128	0.389 E−13	0.146 E−14	0.10	0.36	2.96
256	0.577 E−13	0.204 E−13	0.24	0.76	23.6
512	0.673 E−13	0.325 E−13	0.47	1.61	189
1024	0.118 E−12	0.817 E−13	0.97	3.54	1512 (est.)
2048	0.190 E−12	0.134 E−12	1.86	7.73	12096 (est.)
4096	0.429 E−12	0.288 E−12	3.93	18.21	96768 (est.)

Example 6. Here we consider the variant of Problem 5 which was described in §4.3. In this example, the numbers $\{x_j\}$ were defined by the formula

$$x_j = -\pi + 2\pi \cdot \frac{j + 0.5 + \delta_j}{N+1} \tag{102}$$

for $j = 0, \dots, N$, where δ_j were randomly distributed on the interval $[-0.1, 0.1]$. In addition, the numbers $\{\beta_{-N/4}, \dots, \beta_{N/4}\}$ were generated randomly on the unit square in the complex plane defined by the formulae

$$0 \le \mathrm{Re}(z) \le 1, \qquad 0 \le \mathrm{Im}(z) \le 1, \tag{103}$$

and the numbers $\{g_0, \dots, g_N\}$ were computed directly in double precision arithmetic according to the formula

$$g_j = \sum_{k=-N/4}^{N/4} \beta_k \cdot e^{ikx_j}. \tag{104}$$

The vector g was then used as input for Algorithm 6. Results of this experiment are presented in Tables 6(a) and 6(b).

The following observations can be made from Tables 1–6 above and are in agreement with results of our more extensive experiments.

1. The errors produced by Algorithms 1, 2, and 3 are comparable with those produced by the corresponding direct methods.

TABLE 6(a)
Example 6. *Single precision computations.*

N	Errors		Timings (sec.)		
	E_∞	E_2	Alg. init.	Alg. eval.	Direct
64	0.195 E−05	0.174 E−05	0.09	0.004	0.36
128	0.412 E−05	0.286 E−05	0.17	0.007	2.78
256	0.105 E−04	0.913 E−05	0.34	0.014	23.0
512	0.195 E−04	0.147 E−04	0.70	0.031	184
1024	0.474 E−04	0.320 E−04	1.41	0.066	1472 (est.)
2048	0.836 E−04	0.631 E−04	3.05	0.147	11776 (est.)
4096	0.173 E−03	0.135 E−03	7.22	0.330	94208 (est.)

TABLE 6(b)
Example 6. *Double precision computations.*

N	Errors		Timings (sec.)		
	E_∞	E_2	Alg. init.	Alg. eval.	Direct
64	0.878 E−14	0.762 E−14	0.32	0.008	0.37
128	0.102 E−13	0.981 E−14	0.62	0.018	2.96
256	0.213 E−13	0.180 E−13	1.25	0.039	23.6
512	0.444 E−13	0.376 E−13	2.64	0.083	189
1024	0.679 E−13	0.520 E−13	5.82	0.170	1512 (est.)
2048	0.151 E−12	0.115 E−12	12.37	0.359	12096 (est.)
4096	0.308 E−12	0.232 E−12	27.61	0.766	96768 (est.)

2. The timings for Algorithms 1 and 2 are similar, which is to be expected since Problem 2 is the adjoint of Problem 1. Algorithm 3 is about twice as costly, which is in agreement with the fact that it is a synthesis of Algorithms 1 and 2.

3. In single precision computations for this particular architecture, implementation and range of N, Algorithms 1 and 2 are less than four times as costly as an FFT of the same size. For double precision the ratio is roughly six. These ratios decrease as N increases.

4. The extrapolated break-even point of Algorithms 1 and 2 is at roughly $N = 32$ if the initialization time is ignored. If the initialization time is included, the break-even point is at $N = 256$. For Algorithm 3, the break-even points are at $N = 8$ without initialization and at $N = 32$ with initialization.

5. The timings for Algorithms 4 and 5 are similar as expected, since Problem 5 is the adjoint of Problem 4.

6. The break-even points of Algorithms 4 and 5 are at roughly $N = 32$. For Algorithm 6, the break-even points are at $N = 32$ if the initialization time is taken into account and at $N = 16$ if it is ignored.

7. Algorithms 1 and 2 tend to be slightly more accurate than their inverses, Algorithms 4 and 5.

8. The initialization for Algorithm 6 is computationally costly, but subsequent evaluations require much less CPU time than evaluations for Algorithm 5.

Remark 7.4. The CPU timings for Algorithms 1, 2, and 3 are independent of the particular distributions of ω and x, whereas the timings for Algorithms 4 and 5 are sensitive to the distributions of these vectors.

8. Generalizations and conclusions. The results of this paper can be generalized in the following ways.

1. Simple modifications to Algorithms 1, 2, and 3 will allow the efficient application of the linear transformations $F_1, G_1, H_1 : \mathbf{C}^{N+1} \to \mathbf{C}^{M+1}$ defined by

$$F_1(\alpha)_j = \sum_{k=0}^{N} \alpha_k \cdot e^{i\omega_k \cdot 2\pi j/M} \quad \text{for } j = -\frac{M}{2}, \dots, \frac{M}{2}, \tag{105}$$

$$G_1(\beta)_j = \sum_{k=-N/2}^{N/2} \beta_k \cdot e^{ikx_j} \quad \text{for } j = 0, \dots, M, \tag{106}$$

$$H_1(\gamma)_j = \sum_{k=0}^{N} \gamma_k \cdot e^{i\omega_k x_j} \quad \text{for } j = 0, \dots, M. \tag{107}$$

These changes have been implemented.

2. The algorithms of this paper also assume that $\omega_k \in [-N/2, N/2]$ and $x_j \in [-\pi, \pi]$. Other distributions can be handled by appropriately partitioning the vectors ω and x, treating each partition separately and finally combining the results. The following observation describes translation operators which can be used for each partition, in combination with one of Algorithms 1, 2, or 3.

Observation 8.1. Let $a, b > 0, c, d > 0$ be real numbers and suppose that $\omega_k \in [a-b, a+b]$ for $k = 0, \dots, N$ and $x_j \in [c-d, c+d]$ for $j = 0, \dots, M$. Then we can write

$$\sum_{k=0}^{N} \alpha_k \cdot e^{i\omega_k x_j} = e^{iax_j} \cdot \sum_{k=0}^{N} \alpha_k \cdot e^{i(\omega_k - a)c} \cdot e^{i(\omega_k - a)(x_j - c)} \tag{108}$$

$$= e^{iax_j} \cdot \sum_{k=0}^{N} \alpha'_k \cdot e^{i\omega'_k x'_j}, \tag{109}$$

where

$$\begin{aligned} \alpha'_k &= \alpha_k e^{i(\omega_k - a)c}, \\ \omega'_k &= (\omega_k - a)d/\pi, \\ x'_j &= (x_j - c)\pi/d \in [-\pi, \pi]. \end{aligned} \tag{110}$$

Remark 8.2. Such an algorithm will perform efficiently when the points within a partition are close together and there are very few partitions and not so efficiently if the points are widely separated and there are many partitions. Most cases likely to be encountered in practice fall in the former category.

3. The algorithms of this paper are based on a special case of a more general idea, namely, the adaptive use of interpolation techniques to speed up large scale computations. Other examples of this approach include the use of wavelets for the construction of fast numerical algorithms (see, for example, [1], [3]) and the use of multipole or Chebyshev expansions for the compression of certain classes of linear operators (see, for example, [5], [11]).

4. A paper describing a set of algorithms based on a different interpolation technique is currently in preparation.

5. One of the more far-reaching extensions of the results of this paper is a set of algorithms for higher dimensional discrete Fourier transforms. Investigations into this are currently in progress.

6. The Helmholtz equation in two dimensions is given by

$$\nabla^2\phi + \kappa^2\phi = 0, \tag{111}$$

and has particular solutions of the form

$$\phi(x,y) = e^{i\mu x} \cdot e^{i\nu y}, \tag{112}$$

where $\mu^2 + \nu^2 = \kappa^2$. Solutions of this equation consist of linear combinations of such functions that satisfy a set of boundary conditions, and the results of this paper admit a generalization that constitutes a fast Helmholtz solver.

In conclusion, a group of algorithms has been presented for the rapid application and inversion of matrices of the Fourier kernel. These problems can be viewed as generalizations of the discrete Fourier transform, and the algorithms, while making use of certain simple results from analysis, are very versatile and have wide-ranging potential applications in many branches of mathematics, science, and engineering.

Appendix. In this Appendix we present numerical estimates of error bounds for Theorem 2.10 of §2.2. For our experiments we chose $c = 0$ and $d = \pi$ and chose the two sets of values of m, b, and q, which are used in our single and double precision implementations of the algorithms of this paper. The expression

$$r(x) = 1 - e^{bx^2} \cdot \sum_{k=-q/2}^{q/2} \rho_k \cdot e^{ikx}, \tag{113}$$

where ρ_k is defined by (18), was evaluated at $n = 1000$ equally spaced nodes $\{x_k\}$ in the interval $[-\frac{\pi}{m}, \frac{\pi}{m}]$, and the following three quantities were computed:

- the maximum absolute error E_∞ defined by the formula

$$E_\infty = \max_{1\le k\le n} |r(x_k)|, \tag{114}$$

- the relative L_2 error E_2 defined by the formula

$$E_2 = \sqrt{\frac{\sum_{k=1}^{n} |r(x_k)|^2}{n}}, \tag{115}$$

- the error bound E_B of Theorem 2.10 defined by the formula

$$E_B = e^{-b\pi^2(1-1/m^2)} \cdot (4b+9). \tag{116}$$

The results of these two experiments are presented in Table 7.

TABLE 7

m	b	q	E_∞	E_2	E_B
2	0.5993	10	0.825 E−05	0.176 E−05	0.135 E−00
2	1.5629	28	0.400 E−13	0.580 E−14	0.163 E−03

We observe from Table 7 that the error bound E_B of Theorem 2.10 is very weak compared with the experimentally obtained bounds. Indeed, the requirement that E_B be appropriately small would impose much larger values of b and q than are actually needed for the algorithms.

REFERENCES

[1] B. Alpert, G. Beylkin, R. Coifman, and V. Rokhlin, *Wavelet-like bases for the fast solution of second-kind integral equations*, SIAM J. Sci. Comput., 14 (1993) pp. 159–184.

[2] D. H. Bailey and P. N. Swarztrauber, *The fractional fast Fourier transform and applications*, SIAM Rev., 33 (1991), pp. 389–404.

[3] G. Beylkin, R. Coifman, and V. Rokhlin, *Fast wavelet transforms and numerical algorithms* I, Comm. Pure Appl. Math., 44 (1991), pp. 141–183.

[4] E. O. Brigham, *The Fast Fourier Transform and its Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[5] 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.

[6] J. W. Cooley and J. W. Tukey, *An algorithm for the machine computation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.

[7] G. Dahlquist and A. Bjorck, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[8] A. Dutt, *A Fast Algorithm for the Evaluation of Trigonometric Series*, Tech. Rep. 841, Yale Computer Science Dept., Yale Univ., New Haven, CT, 1991.

[9] D. Gottlieb, M. Y. Hussaini, and S. Orszag, in *Spectral Methods for Partial Differential Equations*, R. G. Voigt, D. Gottlieb, and M. Y. Hussaini, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984, p. 1.

[10] I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series and Products*, Academic Press, New York, 1980.

[11] V. Rokhlin, *A fast algorithm for the discrete Laplace transformation*, J. Complexity, 4 (1988), pp. 12–32.

[12] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[13] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[14] H. J. Weaver, *Theory of Discrete and Continuous Fourier Analysis*, Wiley, New York, 1989.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1394–1414, November 1993

NUMERICAL SOLUTION OF THE RIEMANN PROBLEM FOR TWO-DIMENSIONAL GAS DYNAMICS*

CARSTEN W. SCHULZ-RINNE[†], JAMES P. COLLINS[‡], AND HARLAND M. GLAZ[§]

Abstract. The Riemann problem for two-dimensional gas dynamics with isentropic or polytropic gas is considered. The initial data is constant in each quadrant and chosen so that only a rarefaction wave, shock wave, or slip line connects two neighboring constant initial states. With this restriction sixteen (respectively, fifteen) genuinely different wave combinations for isentropic (respectively, polytropic) gas exist. For each configuration the numerical solution is analyzed and illustrated by contour plots. Additionally, the required relations for the initial data and the symmetry properties of the solutions are given. The chosen calculations correspond closely to the cases studied by T. Zhang and Y. Zheng [*SIAM J. Math. Anal.*, 21 (1990), pp. 593–630], so that the analytical theory can be directly compared to our numerical study.

Key words. Riemann problem, gas dynamics, Godunov method, wave interaction

AMS subject classifications. 35A40, 35L65, 35L67, 65M06, 76N15

1. Introduction. The interaction of elementary waves for systems of hyperbolic conservation laws is the key mechanism in determining the qualitative properties of solutions to more general initial value problems. For the genuinely nonlinear case and one space dimension, these basic interactions along with the Riemann problem can be worked out analytically, at least for small enough jumps in wave strength [16], and are often solvable for large jumps as well [3], [18]. This knowledge is the basic building block in using techniques such as the random choice method (RCM) to obtain existence results [8] as well as information about the qualitative behavior of solutions [12], and this is especially the case for certain systems that are derived from physical principles, e.g., gas dynamics. Indeed, one can go further and conjecture that RCM, with enough mesh points and sufficient accuracy in the Riemann problem solution, will almost always produce a very close approximation to the exact solution for scalar conservation laws, gas dynamics, and certain other systems in the one-dimensional planar case.

It is natural to try to use elementary waves and Riemann problems as building blocks in constructing solutions in two and three dimensions since the idea is so successful in one dimension. For the scalar case, a great deal of work has been done [10], [11], [17], [19], and much is known about the analytic structure of solutions. However, even here it seems difficult to work in analogy to the methods of [8] and [12] and obtain deep results for general data; also, finite difference schemes based directly on two-dimensional Riemann problem solutions have not appeared.

For gas dynamics and two space dimensions, the situation can be expected to be quite complex and is certainly difficult; very few analytic results are available. Our approach here is to use a high-resolution finite difference scheme as an experimental tool and to try to catalogue the phenomenology of the two-dimensional gas dynamic Riemann problem for a restricted set of initial data; our choices have been strongly influenced by

*Received by the editors March 17, 1992; accepted for publication (in revised form) November 19, 1992.

[†]Seminar für Angewandte Mathematik, ETH Zürich, 8092 Zürich, Switzerland. The support of the German National Scholarship Foundation is gratefully acknowledged.

[‡]Information and Mathematical Sciences Branch, Naval Surface Warfare Center, White Oak, Silver Spring, Maryland 20903. The support of the Defense Nuclear Agency, MIPR 90–567, is gratefully acknowledged.

[§]Department of Mathematics, University of Maryland, College Park, Maryland 20742. The support of the National Science Foundation grant DMS-8703971, of the Applied Mathematical Sciences Program of the U.S. Department of Energy at the Courant Institute of Mathematical Sciences grant DE–FG02–88ER25053, and of the Defense Nuclear Agency contract DNA001–87–0303 is gratefully acknowledged.

the conjectures reported in [20]. A similar approach using the front tracking method is also possible [9], but our results here indicate that such an effort would be awkward to implement. However, the set of results obtained here, suitably augmented to account for a larger data set, might be useful in constructing numerical methods along these lines by supplying the appropriate jump conditions at solution singularities.

The most important feature of elementary waves and Riemann problems is that their solutions may be assumed to be self-similar; this reduces the number of dimensions by one and leads to the analytic theory for the one-dimensional case. The dimensional reduction in two-dimensions is from three to two dimensions and the resulting system is still a partial differential equation (PDE). A set of interesting self-similar data for gas dynamics, from a historical point of view, is the problem of oblique shock wave reflection, which can be experimentally simulated by situating a wedge in a shock tube downstream of the incident shock. Experimental, analytical, and numerical investigations of this problem go back to Mach and von Neumann; the extensive literature is discussed in [5]. The numerical scheme used to obtain the results in our work here has been implemented for this configuration as well [6], [7]. It has been conjectured [20], and will be shown here (by computational experiments), that at least some of these results also arise from two-dimensional Riemann problems.

The collection of self-similar data in two-dimensions is very diverse, as already illustrated by the possible wedge configurations and their solutions. Actually, any collection of rays centered at some point (always the origin here) and separated by constant states will lead to a self-similar solution. Our study here is restricted to the special case of four constant states, one in each quadrant. We impose a further restriction by insisting that the waves separating each pair of adjoining states be a single elementary wave; this corresponds to the set of conjectures proposed in [20], which partially inspired the work here.

We remark that even the case of a single planar shock wave situated in $\mathbb{R}^2$ is not necessarily trivial, at least numerically. Indeed, certain conditions must be satisfied by the data and the equation of state in order that the wave be stable [13], [14]. However, stability is guaranteed for the genuinely nonlinear case and our work so far has been for polytropic gases.

All computations are performed using the second-order Eulerian Godunov method as formulated in [1] along with the dissipative mechanisms discussed in [2]. The capabilities of this scheme applied to a wide variety of applications are summarized in [4]. The boundary conditions are trivial for this study, of course. The initial data is laid down on the grid with the jumps at (the Cartesian) grid interfaces. The calculations are started smoothly by taking the initial Courant–Friedrichs–Lewy (CFL) constraint to be very severe and gradually relaxing it. The (x, y)-plane is covered with a uniform mesh containing 400 cells in each direction for all problems reported here. The calculations were performed on Sun workstations at the ETH Zurich.

The equations of motion, the Riemann problem initial data, and the resulting self-similar equations are discussed in §2; this section also contains a summary of the classification based on [20] and derived in [15].

In §3 the numerical solutions and their structures are presented and analyzed. Through observation of the time-dependent solution, it is clear that our mesh is sufficient for the solutions to relax to their pseudosteady states; thus, our illustrations may be assumed to be situated in the (ξ, η)-plane. A few computations exhibit an unsteady Kelvin–Helmholtz instability across slip surfaces. For these cases, we cannot assume the existence of a true pseudosteady state although the overall structures are stable in time;

see [5] for further discussion. The results are illustrated by contour plots of the density and self-similar Mach number. In the (ξ, η)-plane the pseudostationary flow is transonic. The four elementary waves, extending from the boundary of the domain parallel to the coordinate axes, interact and/or reach the region of subsonic flow. In the cases with four shock waves, the interaction of the shock fronts results in simple, complex, or double Mach reflections depending on the initial strength of the waves. Four slip lines lead to large-scale vortex creation in one case. With some wave combinations the region of subsonic flow is partly bounded by newly created shocks. Our conclusions are presented in §4.

2. Problem formulation and classification. The Riemann problem for gas dynamics in two space dimensions, both for isentropic and polytropic gas, is formulated as follows. We start with the Euler equations of inviscid compressible isentropic flow consisting of the equations for conservation of mass and momentum. For polytropic gas we have an additional equation for the conservation of energy. The conservation form of these equations in Cartesian coordinates is

$$U_t + F(U)_x + G(U)_y = 0, \tag{1}$$

where

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \end{pmatrix}, \quad F(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \end{pmatrix}, \quad G(U) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \end{pmatrix}$$

for isentropic gas and

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad F(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(\rho E + p) \end{pmatrix}, \quad G(U) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(\rho E + p) \end{pmatrix}$$

for polytropic gas. Here ρ is the density, u the x-velocity component, v the y-velocity component, p the pressure, and E the total specific energy. The system is closed by specifying an equation of state. For isentropic gas we take $p = A\rho^\gamma$, where $A > 0$ and $\gamma > 1$ are constants. For polytropic gas we have instead

$$E = \frac{1}{(\gamma - 1)} \frac{p}{\rho} + \frac{u^2 + v^2}{2},$$

where $\gamma > 1$ is constant. The number γ is called the polytropic exponent and, since it is assumed constant in the present study, may be interpreted physically as the ratio of specific heats for the polytropic case.

The eigenvalues of the Jacobian matrix $\nabla_U F$ (or $\nabla_U G$) are $\lambda_- = u - c$, $\lambda_0 = u$ and $\lambda_+ = u + c$ (or $\lambda_- = v - c$, $\lambda_0 = v$ and $\lambda_+ = v + c$). These are the characteristic speeds for one-dimensional gas dynamics and are needed here only for the implementation of the (operator split) numerical method. The sound speed c is defined by $c^2 = \gamma p/\rho$.

The Riemann problem in the (x, y)-plane is the initial value problem for (1) with initial data

$$(\rho, u, v)(x, y, 0) = (\rho_i, u_i, v_i), \qquad i = 1, \ldots, 4 \tag{2}$$

for isentropic gas and

$$(p, \rho, u, v)(x, y, 0) = (p_i, \rho_i, u_i, v_i), \qquad i = 1, \dots, 4 \tag{2'}$$

for polytropic gas where i denotes the ith quadrant.

The solution is a function of the similarity variables $\xi = x/t$ and $\eta = y/t$. In these new variables, (1) becomes

$$-\xi U_\xi - \eta U_\eta + F(U)_\xi + G(U)_\eta = 0\,. \tag{3}$$

Introducing the so-called pseudovelocities $\tilde{u} = u - \xi$ and $\tilde{v} = v - \eta$, the system (3) is equivalent to

$$F(\tilde{U})_\xi + G(\tilde{U})_\eta + S(\tilde{U}) = 0, \tag{4}$$

where

$$\tilde{U} = \begin{pmatrix} \rho \\ \rho\tilde{u} \\ \rho\tilde{v} \end{pmatrix}, \qquad S(\tilde{U}) = \begin{pmatrix} 2\rho \\ 3\rho\tilde{u} \\ 3\rho\tilde{v} \end{pmatrix}$$

for isentropic gas and

$$\tilde{U} = \begin{pmatrix} \rho \\ \rho\tilde{u} \\ \rho\tilde{v} \\ \rho\tilde{E} \end{pmatrix}, \qquad S(\tilde{U}) = \begin{pmatrix} 2\rho \\ 3\rho\tilde{u} \\ 3\rho\tilde{v} \\ 2(\rho\tilde{E} + p) + \rho(\tilde{u}^2 + \tilde{v}^2) \end{pmatrix}$$

with

$$\tilde{E} = \frac{1}{(\gamma - 1)}\frac{p}{\rho} + \frac{\tilde{u}^2 + \tilde{v}^2}{2}$$

for polytropic gas. The system (4) can be expressed in quasilinear form as

$$V_\xi + AV_\eta + B = 0,$$

where

$$A = \left[\frac{\partial F}{\partial V}\right]^{-1}\left[\frac{\partial G}{\partial V}\right], \qquad B = \left[\frac{\partial F}{\partial V}\right]^{-1} S,$$

and V represents an arbitrary change of coordinates satisfying $\det(A) \neq 0$. Then the eigenvalues of A are

$$\tilde{\lambda}_0 = \frac{\tilde{v}}{\tilde{u}} \quad \text{and} \quad \tilde{\lambda}_\pm = \frac{\tilde{u}\tilde{v} \pm c\sqrt{\tilde{u}^2 + \tilde{v}^2 - c^2}}{\tilde{u}^2 - c^2} = \frac{\tilde{v}^2 - c^2}{\tilde{u}\tilde{v} \mp c\sqrt{\tilde{u}^2 + \tilde{v}^2 - c^2}}.$$

The initial value problem becomes a boundary value problem at infinity:

$$\left.\begin{array}{c} (\rho, u, v)(\xi, \eta) \to (\rho_i, u_i, v_i) \\ \text{or} \\ (p, \rho, u, v)(\xi, \eta) \to (p_i, \rho_i, u_i, v_i) \end{array}\right\} \text{ for } \xi^2 + \eta^2 \to \infty \text{ and } \left\{\begin{array}{l} \xi > 0,\ \eta > 0,\ i = 1 \\ \xi < 0,\ \eta > 0,\ i = 2 \\ \xi < 0,\ \eta < 0,\ i = 3 \\ \xi > 0,\ \eta < 0,\ i = 4. \end{array}\right.$$

The self-similar solution in the (ξ, η)-plane is called pseudostationary flow. Far enough away from the origin the general solution consists of four planar waves, each parallel to one of the coordinate axes, separating the four constant initial states; also, the eigenvalues $\tilde{\lambda}_\pm$ are guaranteed to be real. In general, a planar wave is formed by up to three elementary waves corresponding to the eigenvalues λ_-, λ_0, and λ_+: a backward rarefaction wave $\overleftarrow{R}$ or shock wave $\overleftarrow{S}$, a slip line (respectively, a contact discontinuity) for isentropic (respectively, polytropic) gas[1] J, and a forward rarefaction wave $\overrightarrow{R}$ or shock wave $\overrightarrow{S}$. In our study of the interaction of the four planar waves we restrict ourselves to situations where each planar wave is a single elementary wave.

The Riemann problem is classified according to the combination of the four elementary planar waves used to define it. Thus we assume that the initial data (2) or (2′) are chosen so that only a rarefaction wave, shock wave, or slip line connects two neighboring constant states. Under this assumption it was proved in [15] that sixteen (respectively, fifteen) different configurations for the Riemann problem for isentropic (respectively, polytropic) gas in two space dimensions exist (symmetric or rotated configurations are systematically eliminated).

$4\,R$:	$\overrightarrow{R}_{21}\overrightarrow{R}_{32}\overrightarrow{R}_{34}\overrightarrow{R}_{41}$	$\overrightarrow{R}_{21}\overleftarrow{R}_{32}\overleftarrow{R}_{34}\overrightarrow{R}_{41}$	
$4\,S$:	$\overleftarrow{S}_{21}\overleftarrow{S}_{32}\overleftarrow{S}_{34}\overleftarrow{S}_{41}$	$\overleftarrow{S}_{21}\overrightarrow{S}_{32}\overrightarrow{S}_{34}\overleftarrow{S}_{41}$	
$2\,R+2\,S$:		$\overrightarrow{R}_{21}\overrightarrow{S}_{32}\overleftarrow{R}_{34}\overleftarrow{S}_{41}$	(only for isentropic gas)
$4\,J$:	$\overleftrightarrow{J_{21}\,J_{32}\,J_{34}\,J_{41}}$	$\overrightarrow{J_{21}\,J_{32}\,J_{34}\,J_{41}}$	
$2\,J+2\,R$:	$\overrightarrow{R}_{21}\,J_{32}\,J_{34}\overrightarrow{R}_{41}$	$\overleftarrow{R}_{21}J_{32}\,J_{34}\,\overleftarrow{R}_{41}$	$J_{21}\,\overrightarrow{R}_{32}J_{34}\,\overrightarrow{R}_{41}$
$2\,J+2\,S$:	$\overleftarrow{S}_{21}\,J_{32}\,J_{34}\,\overleftarrow{S}_{41}$	$\overrightarrow{S}_{21}\,J_{32}J_{34}\,\overrightarrow{S}_{41}$	$J_{21}\,\overleftarrow{S}_{32}\,J_{34}\,\overleftarrow{S}_{41}$
$2\,J+R+S$:	$\overrightarrow{R}_{21}\,J_{32}\,J_{34}\,\overleftarrow{S}_{41}$	$\overleftarrow{R}_{21}\,J_{32}\,J_{34}\,\overrightarrow{S}_{41}$	$J_{21}\,\overleftarrow{S}_{32}\,J_{34}\,\overrightarrow{R}_{41}$

In this table and in the following, E_{ij} with $E \in \{J, \overleftarrow{R}, \overrightarrow{R}, \overleftarrow{S}, \overrightarrow{S}\}$ and $i, j \in \{1, 2, 3, 4\}$ denotes an elementary wave E between the ith and jth quadrant. The notation in the $4\,J$ cases serves to distinguish the two possibilities which are explained in the next section.

3. Computational results. In this section we present the numerical solutions to the configurations one by one. For each of them, we give the relations that must be satisfied by the initial data and the symmetry properties of the solution. The formulas for one-dimensional elementary waves between two constant states, which follow from the simple wave and Rankine–Hugoniot relations [3], are stated in [15]. We use the formulas and the abbreviations introduced there; that is, for a given left and right state (denoted by the indices l and r), we define

$$\Phi_{lr} := \frac{2\sqrt{\gamma}}{\gamma - 1}\left(\sqrt{\frac{p_l}{\rho_l}} - \sqrt{\frac{p_r}{\rho_r}}\right), \qquad \Psi_{lr}^2 := \frac{(p_l - p_r)(\rho_l - \rho_r)}{\rho_l \rho_r} \quad (\Psi_{lr} > 0),$$

and

$$\Pi_{lr} := \left(\frac{p_l}{p_r} + \frac{(\gamma-1)}{(\gamma+1)}\right) \Big/ \left(1 + \frac{(\gamma-1)}{(\gamma+1)}\frac{p_l}{p_r}\right).$$

[1]In the following, the use of the term *slip line* always denotes a slip line for isentropic gas, but should be read to include the possibility of a contact discontinuity for polytropic gas.

Furthermore, we describe the main features of the solution in the (ξ, η)-plane. It always contains a subsonic area ($\tilde{M} < 1$) that is separated from the supersonic area ($\tilde{M} > 1$) by the sonic curve ($\tilde{M} = 1$). Here the self-similar Mach number $\tilde{M}$ is defined by $\tilde{M}^2 = (\tilde{u}^2 + \tilde{v}^2)/c^2$. In the supersonic domain the curves given by $d\eta/d\xi = \tilde{\lambda}_\pm$ are called the $\lambda_\pm$ characteristic lines. As in [20] we outline the behavior of these lines inside the rarefaction waves. The numerical solutions are illustrated by contour plots of the density and self-similar Mach number in the (ξ, η)-plane. In the Mach number plots the subsonic contour lines are dashed. For some configurations containing rarefaction waves the $\lambda_{+(-)}$ characteristic lines inside the fan are shown as solid (dashed) lines. The initial data listed in the figure captions, together with the relations given for the corresponding configuration, are always sufficient to uniquely define the solution.

3.1. Cases not involving slip line initial data. *Four rarefaction waves.* In this case the flow is isentropic.

Configuration 1. $\overrightarrow{R}_{21}\overrightarrow{R}_{32}\overrightarrow{R}_{34}\overrightarrow{R}_{41}$ (Figs. 1 and 3(a)). We have

$$p_1 > p_2, \; p_4 > p_3$$

and

$$\begin{array}{llll} u_2 - u_1 = \Phi_{21}, & u_3 - u_4 = \Phi_{34}, & u_3 = u_2, & u_4 = u_1, \\ v_4 - v_1 = \Phi_{41}, & v_3 - v_2 = \Phi_{32}, & v_2 = v_1, & v_3 = v_4. \end{array}$$

This gives the so-called compatibility condition $\Phi_{21} = \Phi_{34}$. For polytropic gas, we must include the following equations:

$$\rho_i/\rho_j = (p_i/p_j)^{1/\gamma} \quad \text{for} \quad (i,j) \in \{(2,1), (3,4), (3,2), (4,1)\}.$$

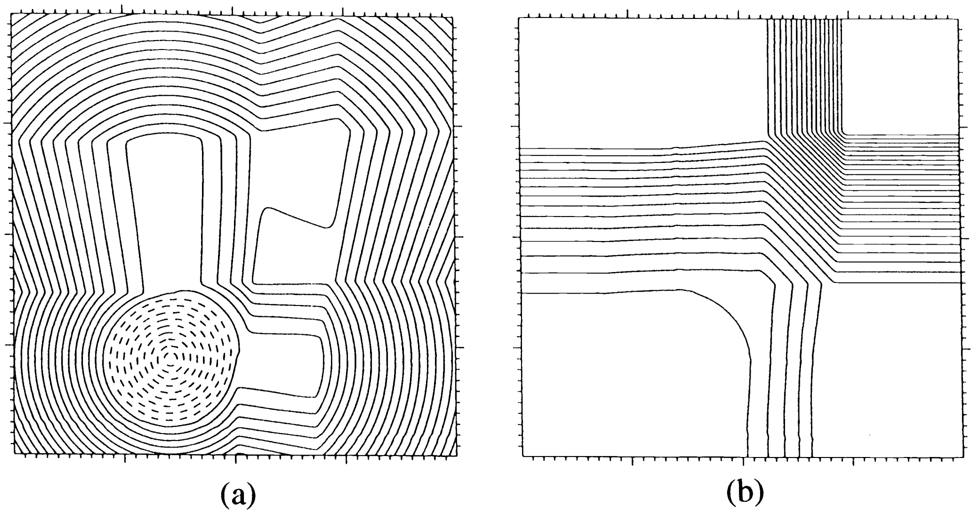


FIG. 1. *Configuration* 1. $p_1 = 1$, $p_2 = .4$, $p_4 = .15$, $\rho_1 = 1$, $u_1 = 0$, $v_1 = 0$. (a) *Self-similar Mach no.* (0.01 *to* 2.99); 29 *contour lines*: 0.10 *to* 2.90 *step* 0.10. (b) *Density* (0.102 *to* 1.000); 30 *contour lines*: 0.11 *to* 0.98 *step* 0.03.

For the case of a vanishing pressure jump in the η-direction, the sonic curve consists of a circle behind the rarefaction fan and a straight line called the sonic stem extending horizontally from the circle to the front of the wave. Inside the rarefaction fan the $\lambda_\pm$ characteristic lines are parallel to the contour lines between the boundary and the sonic stem. There the lines turn downwind to end at the sonic circle.

For the general case, the sonic circles of $\overrightarrow{R}_{32}$ and $\overrightarrow{R}_{34}$ jointly form the subsonic area behind the interaction of the waves. The rarefaction waves $\overrightarrow{R}_{21}$ and $\overrightarrow{R}_{41}$ meet at a point P in the first quadrant. The λ_- characteristic through P forms the lower boundary of the undisturbed $\overrightarrow{R}_{21}$ wave. It crosses $\overrightarrow{R}_{21}$, continues in the second quadrant, turns downward, crosses $\overrightarrow{R}_{32}$, enters the third quadrant, and ends at the sonic curve. Analogously, the λ_+ characteristic through P forms the left boundary of the undisturbed $\overrightarrow{R}_{41}$ wave. It crosses $\overrightarrow{R}_{41}$, continues in the fourth quadrant, turns left, crosses $\overrightarrow{R}_{34}$, and ends at the sonic curve in the third quadrant. The λ_- characteristic lines inside $\overrightarrow{R}_{34}$ are parallel to the contour lines inside the fan and turn downwind to end at the sonic curve. The equivalent is true for the λ_+ characteristic inside $\overrightarrow{R}_{32}$. The traces of the sonic stems in $\overrightarrow{R}_{32}$ and $\overrightarrow{R}_{34}$ are easily found in the Mach number plot (Fig. 1).

For an extensive range of initial data, we found that an intersection of the $\lambda_\pm$ characteristic lines at the lower left side of the subsonic area does not occur; compare [20, Figs. 4.2 and 4.3].

Configuration 2. $\overrightarrow{R}_{21}\overleftarrow{R}_{32}\overleftarrow{R}_{34}\overrightarrow{R}_{41}$ (Figs. 2 and 3(b)). We have

$$p_1 > p_2,\ p_4 < p_3$$

and

$$\begin{array}{llll} u_2 - u_1 = \Phi_{21}, & u_4 - u_3 = \Phi_{34}, & u_3 = u_2, & u_4 = u_1, \\ v_4 - v_1 = \Phi_{41}, & v_2 - v_3 = \Phi_{32}, & v_2 = v_1, & v_3 = v_4 \end{array}$$

so that the compatibility conditions are $\Phi_{21} = -\Phi_{34}$ and $\Phi_{41} = -\Phi_{32}$. For polytropic gas we include the same additional equations as in Configuration 1.

Thus we must have $p_1 = p_3$ and $p_2 = p_4$ implying $u_2 - u_1 = v_4 - v_1$ and $u_4 - u_3 = v_2 - v_3$. Consequently, the solutions are symmetric to $\eta - \xi = v_1 - u_1$ and $\xi + \eta = u_2 + v_2$. Therefore the description focuses on the region $\xi + \eta > u_2 + v_2$ of the domain.

The nonconvex subsonic area lies between the four rarefaction waves. On the segment of the sonic curve facing the first quadrant a shock wave appears. The rarefaction waves $\overrightarrow{R}_{21}$ and $\overrightarrow{R}_{41}$ meet at a point P in the first quadrant. The λ_- characteristic through P forms the lower boundary of the undisturbed $\overrightarrow{R}_{21}$ wave. It crosses $\overrightarrow{R}_{21}$ and turns slightly downward ending at the sonic curve. Analogously, the λ_+ characteristic through P forms the left boundary of the undisturbed $\overrightarrow{R}_{41}$ wave. It crosses $\overrightarrow{R}_{41}$ and turns slightly left ending at the sonic curve. The other $\lambda_\pm$ characteristic lines inside the fans are almost parallel to the corresponding characteristics mentioned above.

The overall structure of our solution corresponds reasonably well with the prediction of [20, Fig. 4.5], although the computed shock on the sonic curve is not present in the prediction. Also, for the calculation presented here, as well as for many others performed by us, the $\lambda_\pm$ characteristic structure never exhibited tangential incidence with the sonic curve and the behavior depicted in [20, Fig. 4.6] was not observed.

Four shock waves.

Configuration 3. $\overleftarrow{S}_{21}\overleftarrow{S}_{32}\overleftarrow{S}_{34}\overleftarrow{S}_{41}$ (Figs. 4 and 5). We have

$$p_1 > p_2,\ p_4 > p_3$$

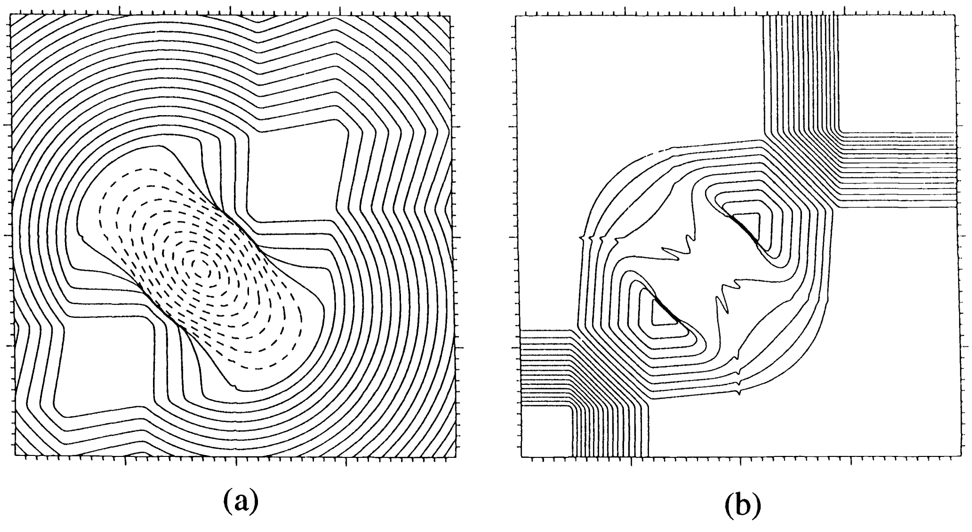


FIG. 2. *Configuration* 2. $p_1 = 1,\ p_2 = .4,\ \rho_1 = 1,\ u_1 = 0,\ v_1 = 0$. (a) *Self-similar Mach no.* (0.01 *to* 2.95); 29 *contour lines*: 0.10 *to* 2.90 *step* 0.10. (b) *Density* (0.258 *to* 1.000); 25 *contour lines*: 0.26 *to* 0.98 *step* 0.03.

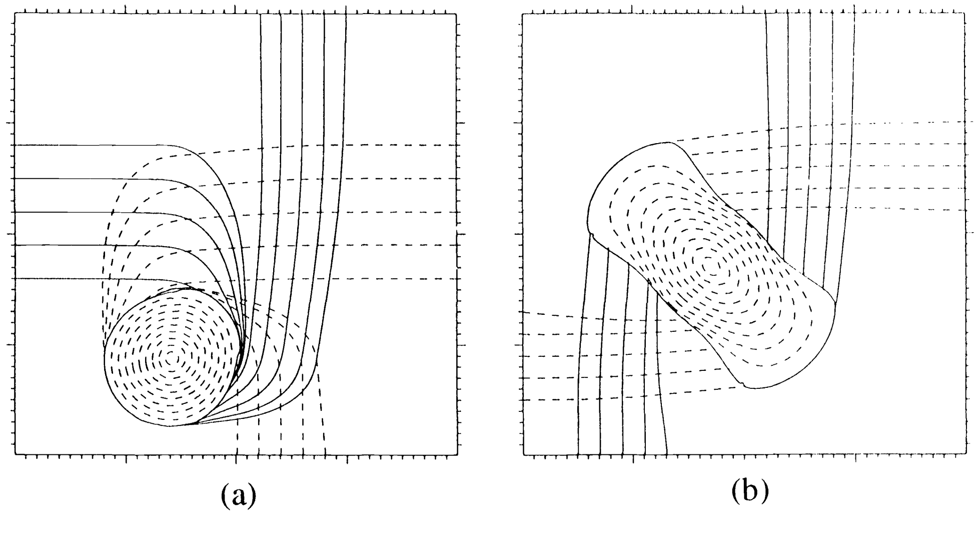


FIG. 3. (a) *Configuration* 1. *Same initial data as in Fig.* 1. *Self-similar Mach no. with char. lines*; 10 *contour lines*: 0.10 *to* 1.00 *step* 0.10. (b) *Configuration* 2. *Same initial data as in Fig. 2. Self-similar Mach no. with char. lines*; 10 *contour lines*: 0.10 *to* 1.00 *step* 0.10.

and

$$u_2 - u_1 = \Psi_{21}, \qquad u_3 - u_4 = \Psi_{34}, \qquad u_3 = u_2, \qquad u_4 = u_1,$$
$$v_4 - v_1 = \Psi_{41}, \qquad v_3 - v_2 = \Psi_{32}, \qquad v_2 = v_1, \qquad v_3 = v_4.$$

This gives the compatibility conditions $\Psi_{21} = \Psi_{34}$ and $\Psi_{41} = \Psi_{32}$. For polytropic gas the following equations are added:

$$\rho_i/\rho_j = \Pi_{ij} \quad \text{for} \quad (i,j) \in \{(2,1),\ (3,4),\ (3,2),\ (4,1)\}.$$

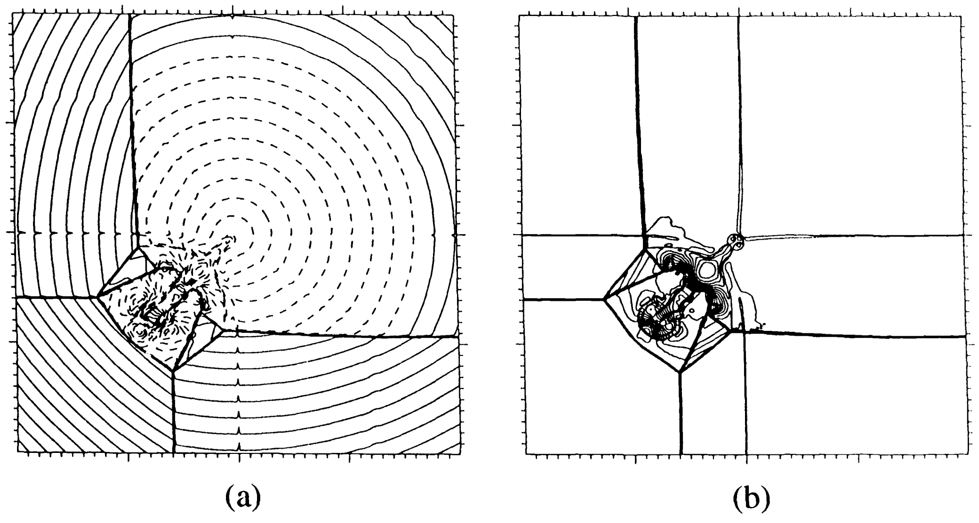


FIG. 4. *Configuration* 3. $p_1 = 1.5$, $p_2 = .3$, $\rho_1 = 1.5$, $u_1 = 0$, $v_1 = 0$. (a) *Self-similar Mach no.* (0.00 *to* 4.67); 46 *contour lines*: 0.10 *to* 4.60 *step* 0.10. (b) *Density* (0.138 *to* 1.756); 32 *contour lines*: 0.16 *to* 1.71 *step* 0.05.

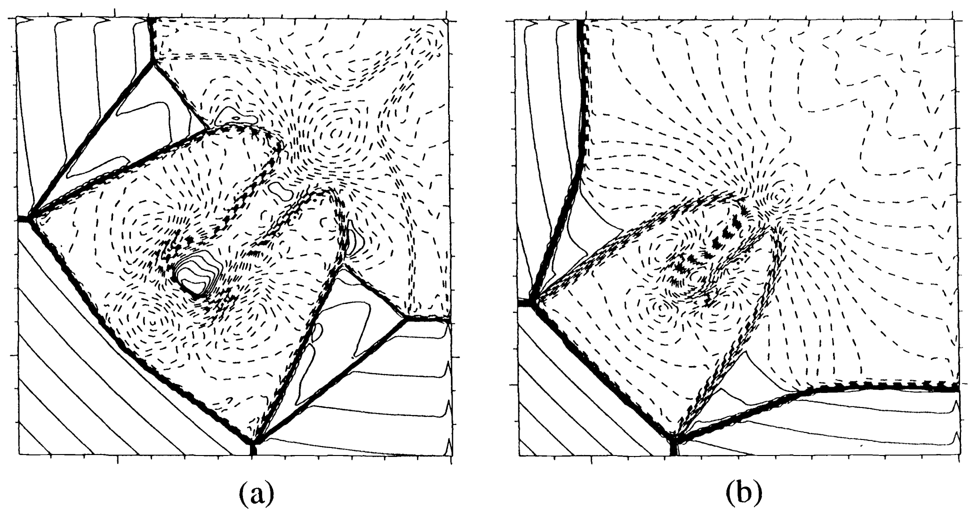


FIG. 5. *Configuration* 3. (a) $p_1 = 1.5$, $p_2 = .3$, $\rho_1 = 1.5$, $u_1 = 0, v_1 = 0$, *showing* 130 × 130 *cells of the upper-right part of the third quadrant. Self-similar Mach no.* (0.01 *to* (3.78); 47 *contour lines*: 0.04 *to* 3.72 *step* 0.08. (b) $p_1 = 1.2$, $p_2 = .35$, $\rho_1 = 1.2$, $u_1 = 0, v_1 = 0$. *showing* 120 × 120 *cells of the upper-right part of the third quadrant. Self-similar Mach no.* (0.00 *to* (3.01); 50 *contour lines*: 0.04 *to* 2.98 *step* 0.06.

It has been proved in [15] that the pressure inequalities required for this configuration and the compatibility conditions listed above can only be satisfied if $p_4 = p_2$ assuming that $1 < \gamma \leq 3$ for polytropic gas. Therefore, we must choose $p_4 = p_2$ (which implies $\rho_4 = \rho_2$), and we get $u_2 - u_1 = v_4 - v_1$. Hence the solutions are symmetric to $\eta - \xi = v_1 - u_1$, and we can concentrate on the region $\eta - \xi > v_1 - u_1$ of the domain.

For the shock speeds σ_{ij} of the shocks $\overleftarrow{S}_{ij}$, the following inequalities hold:

$$\sigma_{21} < \sigma_{34} \quad \text{and} \quad \sigma_{41} < \sigma_{32}.$$

By definition, they are equivalent to

$$u_2 - \sqrt{\frac{\rho_1}{\rho_2}\frac{(p_1 - p_2)}{(\rho_1 - \rho_2)}} < u_3 - \sqrt{\frac{\rho_4}{\rho_3}\frac{(p_4 - p_3)}{(\rho_4 - \rho_3)}}$$

and

$$v_4 - \sqrt{\frac{\rho_1}{\rho_4}\frac{(p_1 - p_4)}{(\rho_1 - \rho_4)}} < v_3 - \sqrt{\frac{\rho_2}{\rho_3}\frac{(p_2 - p_3)}{(\rho_2 - \rho_3)}}.$$

Using $u_3 = u_2$, $v_3 = v_4$, and the compatibility conditions, both inequalities become

$$\rho_1\rho_3 < \rho_2\rho_4 \quad \text{which is equivalent to} \quad p_1p_3 < p_2p_4.$$

Considering the symmetry (for a more general argument not using the assumption $p_2 = p_4$, see [20]), we set

$$p_1 = rp_3 \quad \text{and} \quad p_2 = p_4 = sp_3.$$

Then we must prove that

$$r < s^2 \quad \text{for} \quad r > s > 1$$

under the compatibility condition

$$(r - s)\left(s^{-1/\gamma} - r^{-1/\gamma}\right) = (s - 1)\left(1 - s^{-1/\gamma}\right)$$

for isentropic gas and

$$(r - s)^2 / [(\gamma + 1)r + (\gamma - 1)s] = (s - 1)^2 / [(\gamma + 1) + (\gamma - 1)s]$$

for polytropic gas. The functions

$$f_i(r) = (r - s)\left(s^{-1/\gamma} - r^{-1/\gamma}\right) - (s - 1)\left(1 - s^{-1/\gamma}\right)$$

and

$$f_p(r) = (r - s)^2 [(\gamma + 1) + (\gamma - 1)s] - (s - 1)^2 [(\gamma + 1)r + (\gamma - 1)s]$$

have the following properties:

$$\begin{aligned}
f_i(s) &= -(s - 1)\left(1 - s^{-1/\gamma}\right) < 0,\\
f_i(s^2) &= (s - 1)\left(1 - s^{-1/\gamma}\right)\left(s^{(\gamma-1)/\gamma} - 1\right) > 0,\\
f_i''(r) &= \gamma^{-2} r^{-(2\gamma+1)/\gamma}[(\gamma - 1)r + (\gamma + 1)s] > 0,\\
f_p(s) &= -2\gamma s(s - 1)^2 < 0,\\
f_p(s^2) &= (\gamma - 1)s(s - 1)^2(s^2 - 1) > 0,\\
f_p''(r) &= 2 > 0.
\end{aligned}$$

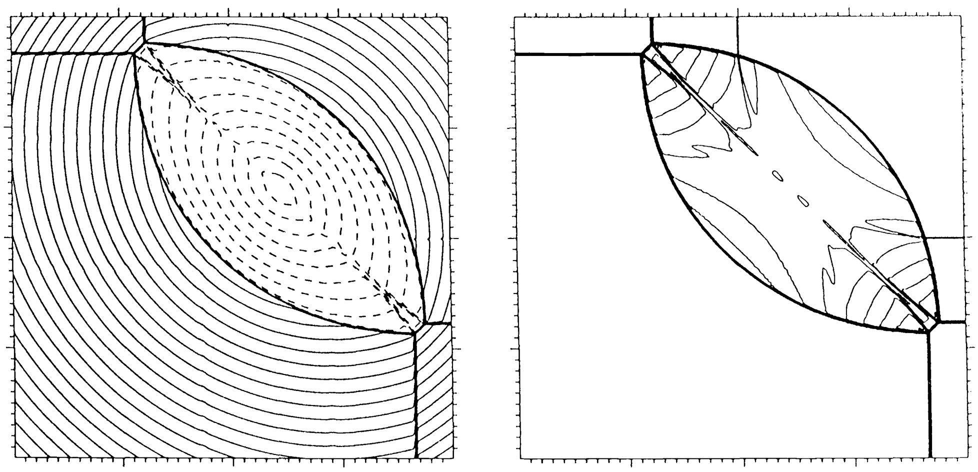
Thus the root $r > s$ of f_i or f_p, which is also the solution of the corresponding compatibility condition, satisfies $r < s^2$.

The shock $\overleftarrow{S}_{21}$ intersects the sonic circle of the constant state in the first quadrant before it meets $\overleftarrow{S}_{32}$ as expected from the shock speed inequalities. $\overleftarrow{S}_{21}$ bends toward the subsonic area. The intersection of $\overleftarrow{S}_{21}$ and $\overleftarrow{S}_{32}$ creates a three-shock configuration. The Mach shock and its symmetric counterpart join and bound the subsonic area. A slip line reaches from the branch point into the subsonic area toward the symmetry axis. Depending on the strength of the shocks we observe the different types of Mach reflections: the simple Mach reflection where the area between the reflected shock and the slip line is subsonic, the complex Mach reflection (Fig. 5(b)) where a part of that area is supersonic, and the double Mach reflection (Figs. 4, 5(a)) with another three-shock configuration. Thus far, we have not observed regular reflection and, for all cases of Mach reflection which we studied, the slip line rolls up into a vortex rather than intersecting the symmetry line at a self-similar stagnation point. The prediction of [20, Fig. 5.2] is remarkably accurate. We note that the extra slip lines at the interfaces of the quadrants in Fig. 4 are numerical artifacts that had been studied earlier [1], [6] and are due to a "starting error."

Configuration 4. $\overleftarrow{S}_{21}\overrightarrow{S}_{32}\overrightarrow{S}_{34}\overleftarrow{S}_{41}$ (Fig. 6) We have

$$p_1 > p_2,\ p_4 < p_3$$

and the same equations and compatibility conditions as in Configuration 3.



(a) (b)

FIG. 6. *Configuration* 4. $p_1 = 1.1$, $p_2 = .35$, $\rho_1 = 1.1$, $u_1 = 0$, $v_1 = 0$. (a) *Self-similar Mach no.* (0.00 *to* 3.39); 33 *contour lines*: 0.10 *to* 3.30 *step* 0.10. (b) *Density* (0.506 *to* 1.932); 29 *contour lines*: 0.52 *to* 1.92 *step* 0.05.

Necessarily, we must have $p_1 = p_3$ and $p_2 = p_4$ (which implies $\rho_1 = \rho_3$ and $\rho_2 = \rho_4$) yielding $u_2 - u_1 = v_4 - v_1$ and $u_3 - u_4 = v_3 - v_2$. Consequently, the solutions are symmetric to $\eta - \xi = v_1 - u_1$ and $\xi + \eta = u_2 + v_2$. Therefore, the description focuses on the region $\eta - \xi > v_1 - u_1$ of the domain.

It is easy to conclude from the equations that $\sigma_{21} < \sigma_{34}$ and $\sigma_{41} < \sigma_{32}$. Accordingly, the shock $\overleftrightarrow{S}_{21}$ interacts with $\overrightarrow{S}_{32}$. This interaction creates a pair of three-shock configurations. The subsonic area is bounded by the joining Mach shocks and the reflected shocks so that it has an oval shape. As in Configuration 3 we observe the different types of Mach reflections determined by the strength of the shocks. Illustrated here is a case of simple Mach reflection.

Two rarefaction and two shock waves.

Configuration 6. $\overrightarrow{R}_{21}\overrightarrow{S}_{32}\overleftarrow{R}_{34}\overleftarrow{S}_{41}$ (Fig. 7). We have

$$p_1 > p_2,\ p_4 < p_3$$

and

$$\begin{aligned} &u_2 - u_1 = \Phi_{21}, && u_4 - u_3 = \Phi_{34}, && u_3 = u_2, && u_4 = u_1, \\ &v_4 - v_1 = \Psi_{41}, && v_3 - v_2 = \Psi_{32}, && v_2 = v_1, && v_3 = v_4, \end{aligned}$$

so that the compatibility conditions are $\Phi_{21} = -\Phi_{34}$ and $\Psi_{41} = \Psi_{32}$.

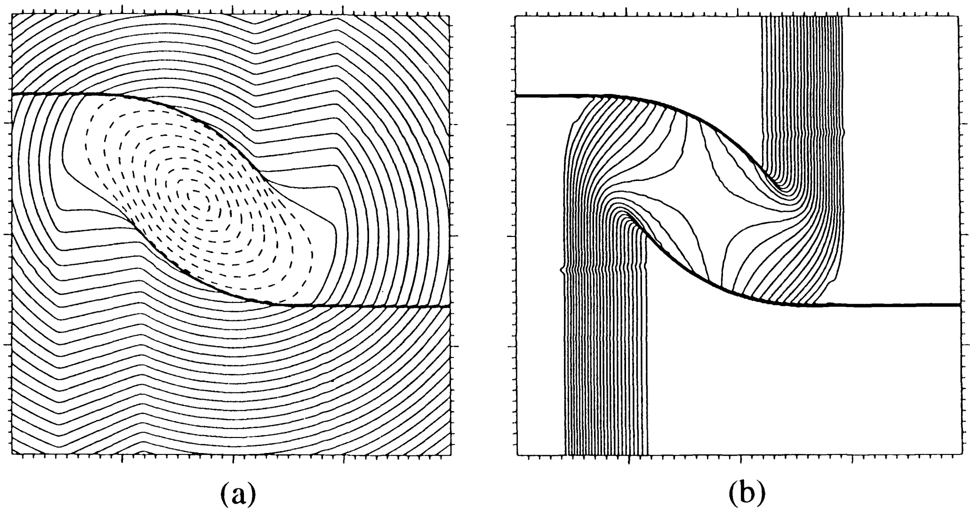


FIG. 7. *Configuration* 6. $A = 1,\ \gamma = 1.4,\ \rho_1 = 1,\ \rho_2 = .5,\ u_1 = 0,\ v_1 = 0$. (a) *Self-similar Mach no.* (0.00 *to* 4.00); 40 *contour lines*: 0.10 *to* 4.00 *step* 0.10. (b) *Density* (0.495 *to* 1.002); 25 *contour lines*: 0.51 *to* 0.99 *step* 0.02.

For polytropic gas, the equations

$$\rho_3/\rho_2 = \Pi_{32}, \quad \rho_4/\rho_1 = \Pi_{41}, \quad \rho_2/\rho_1 = (p_2/p_1)^{1/\gamma}, \quad \text{and} \quad \rho_3/\rho_4 = (p_3/p_4)^{1/\gamma}$$

must be added. As mentioned in §2, this configuration is impossible for this case [15].

For isentropic gas the pressure inequalities required for this configuration and the compatibility conditions listed above can only be satisfied if $p_3 = p_1$ and $p_4 = p_2$ [15]. Consequently, we must choose $p_3 = p_1$ and $p_4 = p_2$ (which implies $\rho_3 = \rho_1$ and $\rho_4 = \rho_2$). Thus the solutions are symmetric with respect to the point $(\xi, \eta) = (u_1, v_1)$, and we can focus the description on the upper part of the domain.

The shock $\overrightarrow{S}_{32}$ bends downward before it dissolves inside the rarefaction fan of $\overrightarrow{R}_{21}$. Thus, the predicted intersection of the shocks $\overrightarrow{S}_{32}$ and $\overleftarrow{S}_{41}$ (see [20, Fig. 6.4]) does not occur in our computations; as a consequence, the calculated results do not contain any slip surfaces in contrast to [20, Fig. 6.4]. The rarefaction wave $\overleftarrow{R}_{34}$ turns right in front of the shock $\overrightarrow{S}_{32}$. The subsonic area is centered about the point (u_1, v_1). The λ_+ characteristic lines inside $\overrightarrow{R}_{21}$ are parallel to the contour lines inside the fan and end at the sonic curve.

3.2. Cases involving slip line initial data. Now we discuss the eleven possible configurations involving slip lines. Beside the relations for the initial data listed for each configuration, we must include additional relations for polytropic gas. For a rarefaction or a shock wave between the ith and jth quadrant ($i, j \in \{1, \dots, 4\}$), we add

$$\rho_i/\rho_j = (p_i/p_j)^{1/\gamma} \quad \text{or} \quad \rho_i/\rho_j = \Pi_{ij},$$

respectively.

The predicted results of [20] are considerably more speculative for these more difficult cases. Consequently, a feature-by-feature comparison with the computed solutions will not be attempted.

Four slip lines.

Configuration A. $\overleftrightarrow{J_{21}J_{32}J_{34}J_{41}}$ (motion in opposite directions; Fig. 8). We have $p_1 = p_2 = p_3 = p_4$ and

$$u_1 = u_2 < u_3 = u_4, \qquad v_1 = v_4 < v_3 = v_2.$$

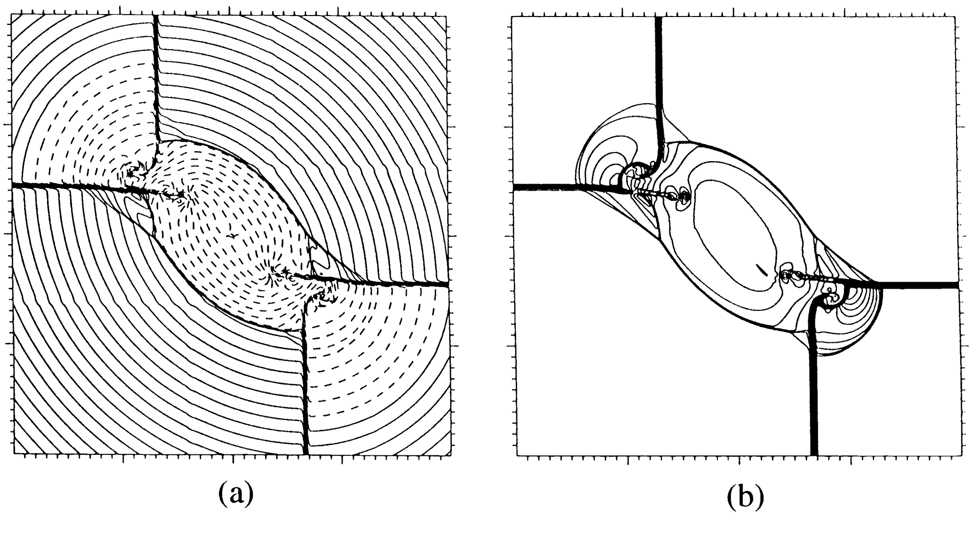


FIG. 8. *Configuration* A. $p_1 = 1$, $\rho_1 = 1$, $\rho_2 = 2$, $\rho_3 = 1$, $\rho_4 = 3$, $u_1 = -.75$, $u_3 = .75$, $v_1 = -.5$, $v_2 = .5$. (a) *Self-similar Mach no.* (0.00 *to* 3.39); 33 *contour lines*: 0.10 *to* 3.30 *step* 0.10. (b) *Density* (1.000 *to* 4.011); 30 *contour lines*: 1.05 *to* 3.95 *step* 0.10.

Only for isentropic gas are the solutions symmetric with respect to the point $(\xi, \eta) = \left(\frac{1}{2}(u_1 + u_3), \frac{1}{2}(v_1 + v_2)\right)$. The slip lines J_{21} and J_{32} meet the sonic circle of the constant state in the second quadrant and continue as almost straight lines so that a quarter of the sonic circle lies in between. The equivalent is true for the slip lines J_{34} and J_{41} in the fourth quadrant. Inside the subsonic area the slip lines bend and end in spirals. Centered about the point $\left(\frac{1}{2}(u_1 + u_3), \frac{1}{2}(v_1 + v_2)\right)$ there is an oval part of the subsonic area bounded by shocks. These shocks form simple Mach reflections near the slip lines (the contact surfaces emanating from the triple points are only barely discernible here).

Configuration B. $\overrightarrow{J_{21}J_{32}J_{34}J_{41}}$ (clockwise motion; Fig. 9). We have $p_1 = p_2 = p_3 = p_4$ and

$$u_1 = u_2 > u_3 = u_4, \quad v_1 = v_4 < v_3 = v_2.$$

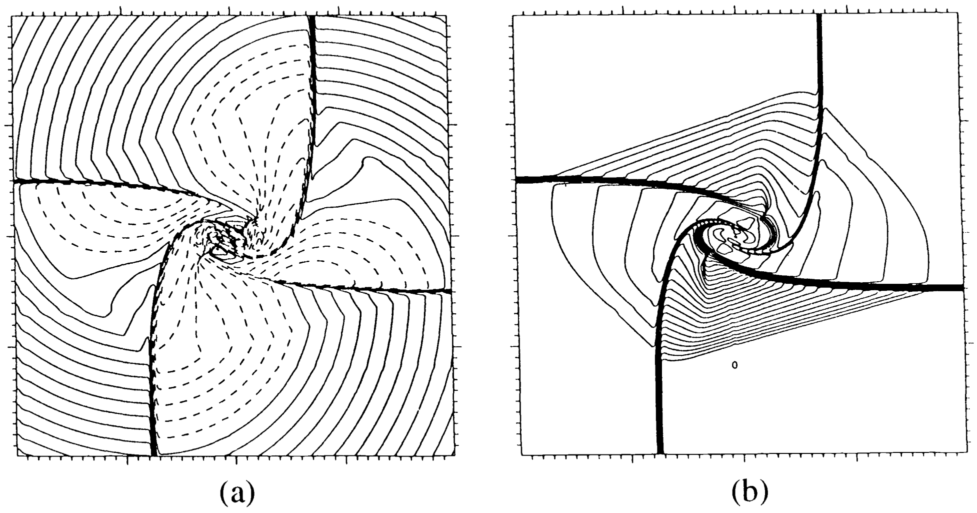


FIG. 9. *Configuration* B. $p_1 = 1$, $\rho_1 = 1$, $\rho_2 = 2$, $\rho_3 = 1$, $\rho_4 = 3$, $u_1 = .75$, ; $u_3 = -.75$, $v_1 = -.5$, $v_2 = .5$. (a) *Self-similar Mach no.* (0.04 *to* 2.35); 23 *contour lines*: 0.10 *to* 2.30 *step* 0.10. (b) *Density* (0.207 *to* 3.064); 29 *contour lines*: 0.25 *to* 3.05 *step* 0.10.

The solutions have the same symmetry properties as in Configuration A. The structure of the solutions is one of a vortex turning clockwise. The slip lines spiral around its center, and the subsonic area has the shape of a four-bladed propeller.

Two neighboring slip lines.

Configuration C. $\overrightarrow{R}_{21}J_{32}J_{34}\overrightarrow{R}_{41}$ (Fig. 10). We have $p_1 > p_2 = p_3 = p_4$ and

$$u_2 - u_1 = \Phi_{21}, \quad u_3 = u_4 = u_1, \quad v_4 - v_1 = \Phi_{41}, \quad v_3 = v_2 = v_1.$$

The solutions are symmetric to $\eta - \xi = v_1 - u_1$. In the region $\xi + \eta > u_3 + v_3$ the solution resembles that of Configuration 2. The slip lines J_{32} and J_{34} meet the sonic circle of the constant state in the third quadrant and continue as almost straight lines so that a quarter of the sonic circle lies in between.

Configuration D. $\overleftarrow{R}_{21}J_{32}J_{34}\overleftarrow{R}_{41}$ (Fig. 11). We have $p_1 < p_2 = p_3 = p_4$ and

$$u_1 - u_2 = \Phi_{21}, \quad u_3 = u_4 = u_1, \quad v_1 - v_4 = \Phi_{41}, \quad v_3 = v_2 = v_1.$$

The solutions are symmetric to $\eta - \xi = v_1 - u_1$. The slip lines J_{32} and J_{34} meet the sonic circle of the constant state in the third quadrant and continue slightly bent so that a quarter of the sonic circle lies in between. The $\lambda_{+(-)}$ characteristic lines inside $\overleftarrow{R}_{41}$ ($\overleftarrow{R}_{21}$) are almost parallel to the contour lines inside the fan and end at the sonic circle. Outside the rarefaction waves the subsonic area is bounded by a circular shock wave.

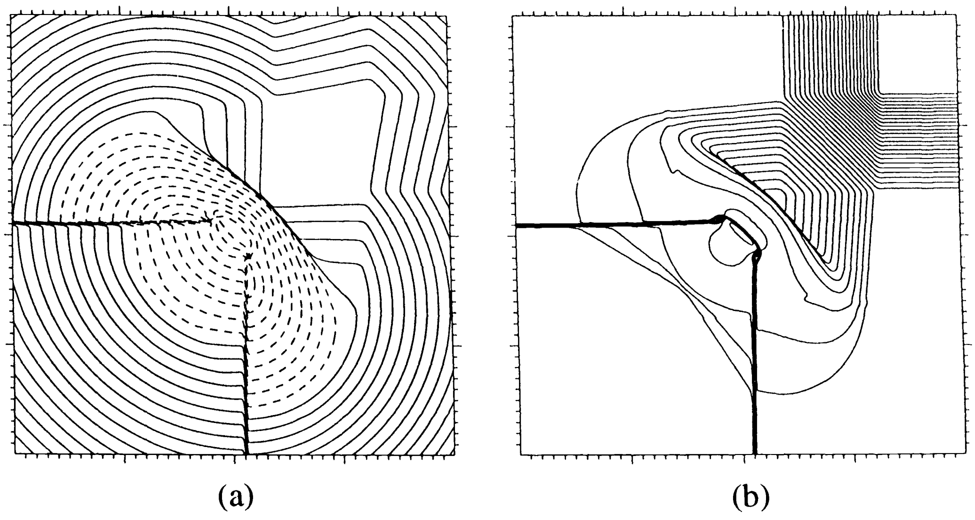


FIG. 10. *Configuration* C. $p_1 = 1,\ p_2 = .4,\ \rho_1 = 1,\ \rho_3 = .8,\ u_1 = .1,\ v_1 = .1$. (a) *Self-similar Mach no.* (0.00 *to* 2.60); 26 *contour lines*: 0.10 *to* 2.60 *step* 0.10. (b) *Density* (0.258 *to* 1.000); 37 *contour lines*: 0.27 *to* 0.99 *step* 0.02.

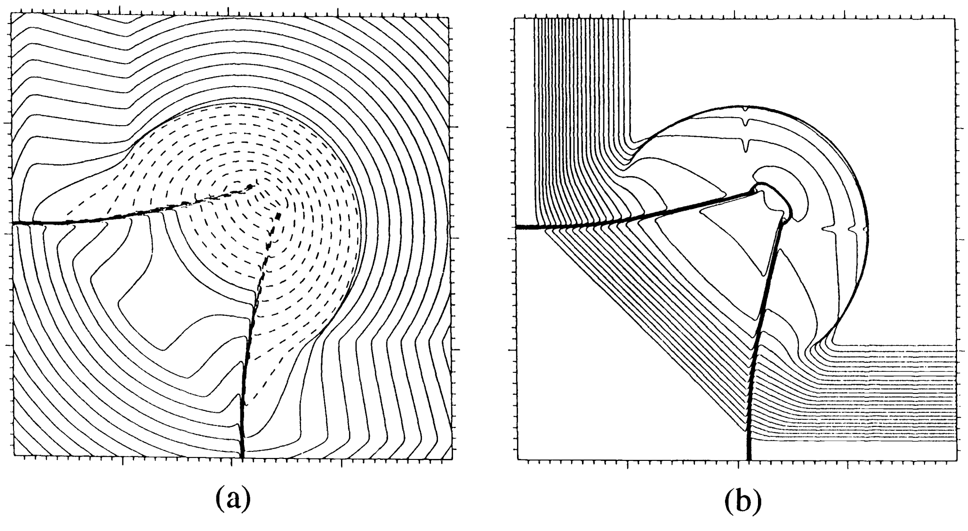


FIG. 11. *Configuration* D. $p_1 = .4,\ p_2 = 1,\ \rho_2 = 1,\ \rho_3 = .8,\ u_1 = .1,\ v_1 = .1$. (a) *Self-similar Mach no.* (0.00 *to* 2.61); 26 *contour lines*: 0.10 *to* 2.60 *step* 0.10. (b) *Density* (0.503 *to* 1.001); 25 *contour lines*: 0.51 *to* 0.99 *step* 0.02.

Configuration E. $\overleftarrow{S}_{21}J_{32}J_{34}\overleftarrow{S}_{41}$ (Fig. 12). We have $p_1 > p_2 = p_3 = p_4$ and

$$u_2 - u_1 = \Psi_{21}, \quad u_3 = u_4 = u_1, \quad v_4 - v_1 = \Psi_{41}, \quad v_3 = v_2 = v_1.$$

The solutions are symmetric to $\eta - \xi = v_1 - u_1$. The shocks $\overleftarrow{S}_{21}$ and $\overleftarrow{S}_{41}$ intersect the sonic circle of the constant state in the first quadrant and end at the slip lines. Between the slip lines J_{32} and J_{34}, which bend inside the subsonic area and end in

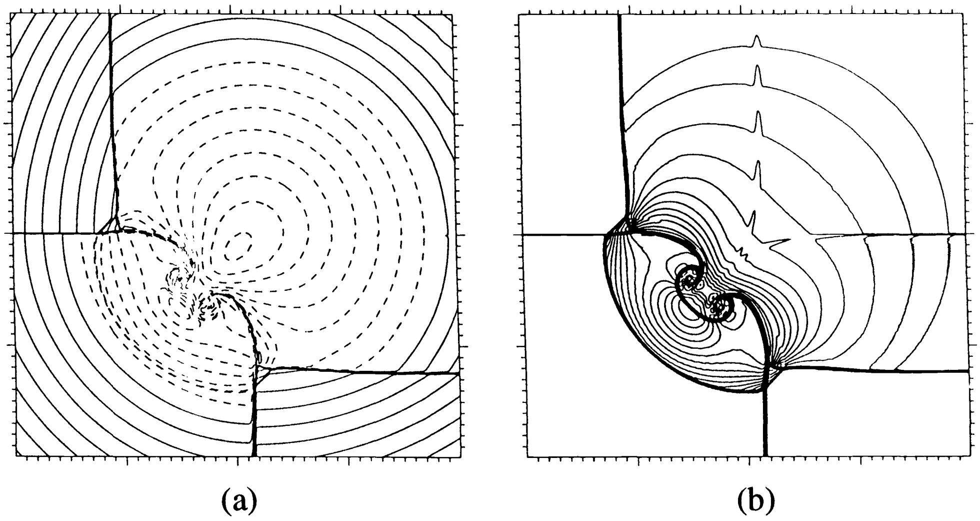


FIG. 12. *Configuration* E. $p_1 = 1,\ p_2 = .4,\ \rho_1 = 1,\ \rho_3 = .8,\ u_1 = .1,\ v_1 = 0$. (a) *Self-similar Mach no.* (0.01 *to* 2.43); 24 *contour lines*: 0.10 *to* 2.40 *step* 0.10. (b) *Density* (0.531 *to* 1.225); 34 *contour lines*: 0.55 *to* 1.21 *step* 0.02.

spirals, the subsonic area is bounded by an oval shock wave. This shock is connected to the shocks $\overleftarrow{S}_{21}$ and $\overleftarrow{S}_{41}$ by two simple Mach reflections.

Configuration F. $\overrightarrow{S}_{21}J_{32}J_{34}\overrightarrow{S}_{41}$ (Fig. 13). We have $p_1 < p_2 = p_3 = p_4$ and the same equations as in Configuration E.

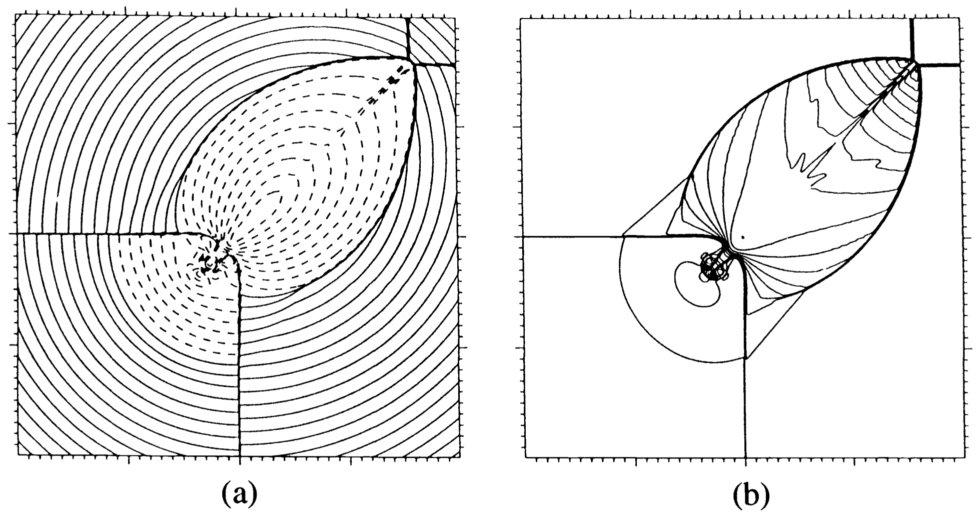
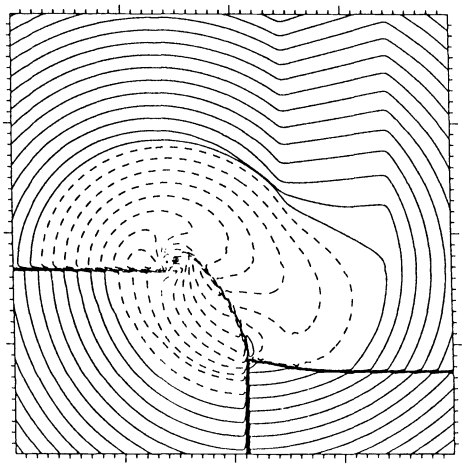


FIG. 13. *Configuration* F. $p_1 = .4,\ p_2 = 1,\ \rho_2 = 1,\ \rho_3 = .8,\ u_1 = 0,\ v_1 = 0$. (a) *Self-similar Mach no.* (0.00 *to* 2.84); 28 *contour lines*: 0.10 *to* 2.80 *step* 0.10. (b) *Density* (0.531 *to* 1.708); 30 *contour lines*: 0.54 *to* 1.70 *step* 0.04.

The solutions are symmetric to $\eta - \xi = v_1 - u_1$. The slip lines J_{32} and J_{34} meet the sonic circle of the constant state in the third quadrant and continue as almost straight lines so that a quarter of the sonic circle lies in between. Inside the subsonic area the slip lines bend and end in spirals. The shocks $\overrightarrow{S}_{21}$ and $\overrightarrow{S}_{41}$ interact like the pair of shocks in Configuration 4.

Configuration G. $\overrightarrow{R}_{21}J_{32}J_{34}\overleftarrow{S}_{41}$ (Fig. 14). We have $p_1 > p_2 = p_3 = p_4$ and

$$u_2 - u_1 = \Phi_{21}, \quad u_3 = u_4 = u_1, \quad v_4 - v_1 = \Psi_{41}, \quad v_3 = v_2 = v_1.$$


(a)

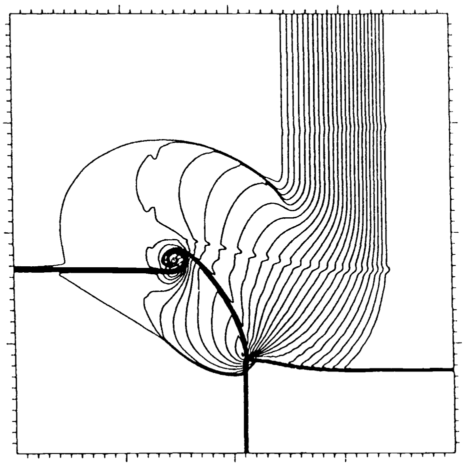

(b)

FIG. 14. *Configuration* G. $p_1 = 1$, $p_2 = .4$, $\rho_1 = 1$, $\rho_3 = .8$, $u_1 = .1$, $v_1 = -.3$. (a) *Self-similar Mach no.* (0.03 *to* 2.69); 26 *contour lines*: 0.10 *to* 2.60 *step* 0.10. (b) *Density* (0.429 *to* 1.004); 29 *contour lines*: 0.43 *to* 0.99 *step* 0.02.

The slip lines J_{32} and J_{34} bend after entering the subsonic area and end in a spiral. The rarefaction wave $\overrightarrow{R}_{21}$ turns backward in front of the shock $\overleftarrow{S}_{41}$ that ends at the slip line J_{34}. The λ_+ characteristic lines inside $\overrightarrow{R}_{21}$ are almost parallel to the contour lines inside the fan and end at the sonic circle.

Configuration H. $\overleftarrow{R}_{21}J_{32}J_{34}\overrightarrow{S}_{41}$ (Fig. 15). We have $p_1 < p_2 = p_3 = p_4$ and

$$u_1 - u_2 = \Phi_{21}, \quad u_3 = u_4 = u_1, \quad v_4 - v_1 = \Psi_{41}, \quad v_3 = v_2 = v_1.$$

The slip lines J_{32} and J_{34} bend after entering the subsonic area and end in a spiral. The shock $\overrightarrow{S}_{41}$ bends downward before it dissolves inside the rarefaction fan of $\overleftarrow{R}_{21}$ that turns right after crossing the slip line J_{32}. The λ_- characteristic lines inside $\overleftarrow{R}_{21}$ are almost parallel to the contour lines inside the fan and end at the sonic circle.

Two nonneighboring slip lines.

Configuration I. $J_{21}\overrightarrow{R}_{32}J_{34}\overrightarrow{R}_{41}$ (Fig. 16). We have $p_1 = p_2 > p_3 = p_4$ and

$$u_1 = u_2 = u_3 = u_4, \quad v_4 - v_1 = \Phi_{41}, \quad v_3 - v_2 = \Phi_{32}.$$

The slip lines J_{21} and J_{34}, which bend and join inside the subsonic area, divide the solution into a left and right part. In general, the rarefaction waves $\overrightarrow{R}_{32}$ and $\overrightarrow{R}_{41}$ move with different velocities so that they are shifted against each other. Below these waves the sonic curve is almost circular. Depending on the vertical velocities, weak shocks are formed below the rarefaction waves.

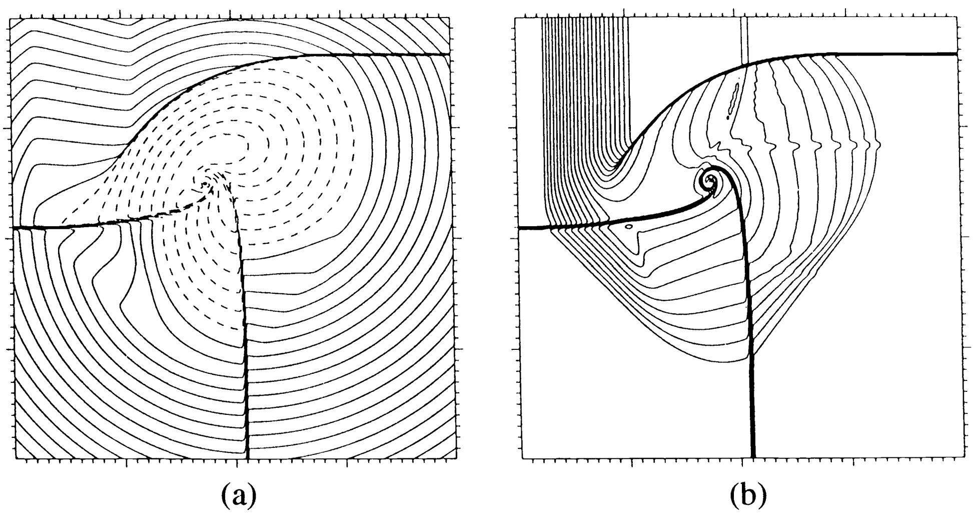


FIG. 15. *Configuration* H. $p_1 = .4$, $p_2 = 1$, $\rho_3 = .8$, $\rho_4 = 1$, $u_1 = .1$, $v_1 = .1$. (a) *Self-similar Mach no.* (0.06 *to* 2.87); 28 *contour lines*: 0.10 *to* 2.80 *step* 0.10. (b) *Density* (0.524 *to* 1.023); 24 *contour lines*: 0.53 *to* 0.99 *step* 0.02.

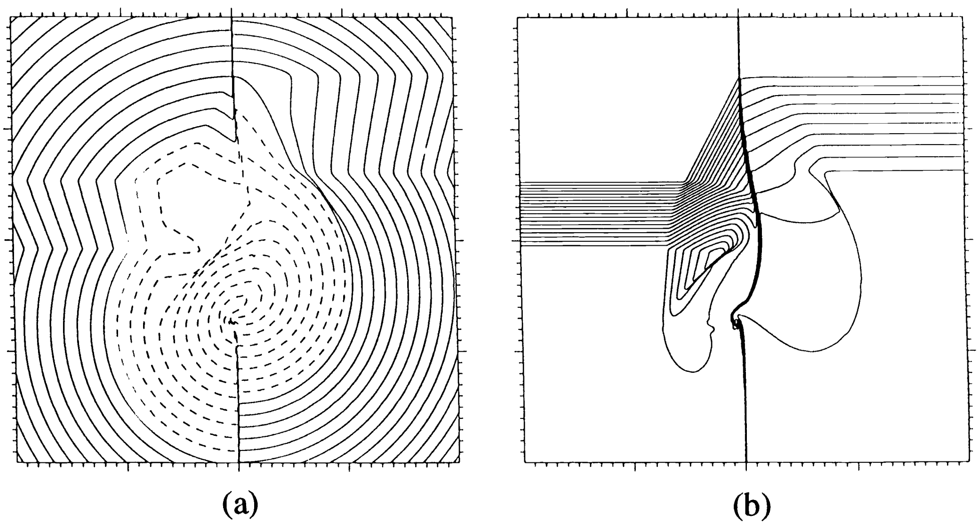


FIG. 16. *Configuration* I. $p_1 = 1$, $p_3 = .4$, $\rho_1 = 1$, $\rho_2 = 2$, $u_1 = 0$, $v_1 = .3$, $v_2 = -.3$. (a) *Self-similar Mach no.* (0.02 *to* 2.44); 24 *contour lines*: 0.10 *to* 2.40 *step* 0.10. (b) *Density* (0.515 *to* 2.000); 30 *contour lines*: 0.53 *to* 1.98 *step* 0.05.

Configuration J. $J_{21}\overleftarrow{S}_{32}J_{34}\overleftarrow{S}_{41}$ (Fig. 17). We have $p_1 = p_2 > p_3 = p_4$ and

$$u_1 = u_2 = u_3 = u_4, \quad v_4 - v_1 = \Psi_{41}, \quad v_3 - v_2 = \Psi_{32}.$$

The slip lines J_{21} and J_{34} separate the solution into a left and right section. Typically, the shocks $\overleftarrow{S}_{32}$ and $\overleftarrow{S}_{41}$ are shifted against each other. Above these shocks the sonic curve is nearly circular. $\overleftarrow{S}_{32}$ and $\overleftarrow{S}_{41}$ are connected by two simple Mach reflections. Their slip lines as well as J_{21} and J_{34}, meet in a vortex inside the subsonic area.

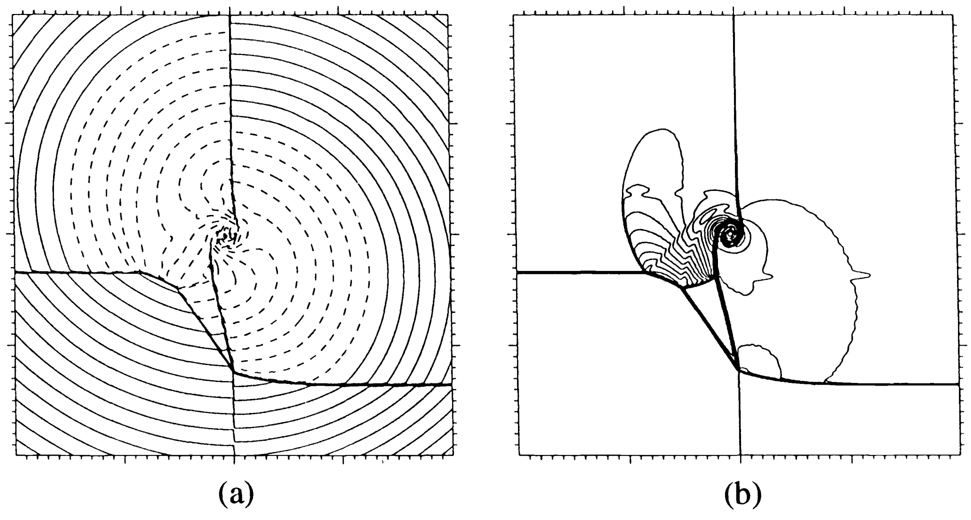


FIG. 17. *Configuration* J. $p_1 = 1$, $p_3 = .4$, $\rho_1 = 1$, $\rho_2 = 2$, $u_1 = 0$, $v_1 = -.3$, $v_2 = .3$. (a) *Self-similar Mach no.* (0.04 *to* 2.55); 25 *contour lines*: 0.10 *to* 2.50 *step* 0.10. (b) *Density* (0.531 *to* 2.407); 37 *contour lines*: 0.57 *to* 2.37 *step* 0.05.

Configuration K. $J_{21}\overleftarrow{S}_{32}J_{34}\overrightarrow{R}_{41}$ (Fig. 18). We have $p_1 = p_2 > p_3 = p_4$ and

$$u_1 = u_2 = u_3 = u_4, \quad v_4 - v_1 = \Phi_{41}, \quad v_3 - v_2 = \Psi_{32}.$$

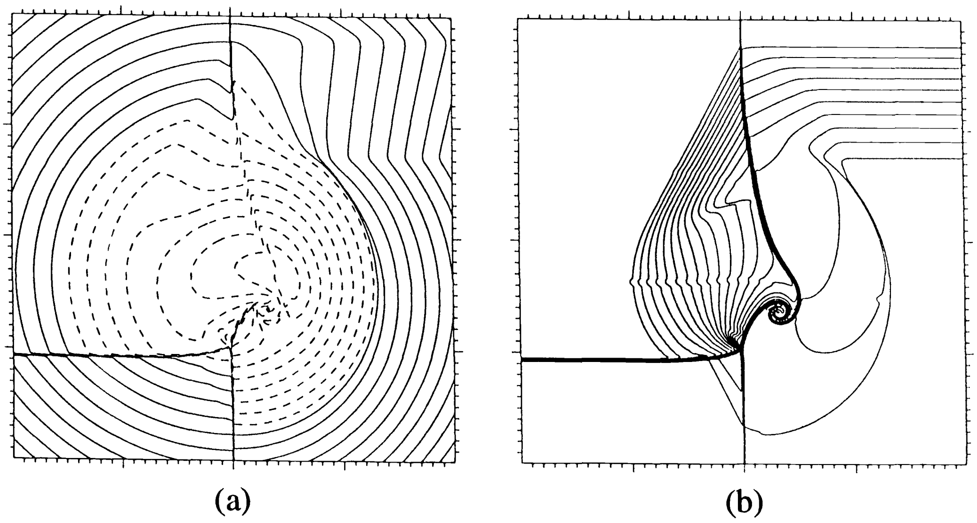


FIG. 18. *Configuration* K. $p_1 = 1$, $p_3 = .4$, $\rho_1 = 1$, $\rho_2 = 2$, $u_1 = 0$, $v_1 = .3$, $v_2 = -.3$. (a) *Self-similar Mach no.* (0.01 *to* 2.09); 20 *contour lines*: 0.10 *to* 2.00 *step* 0.10. (b) *Density* (0.512 *to* 2.003); 30 *contour lines*: 0.53 *to* 1.98 *step* 0.05.

The slip lines J_{21} and J_{34}, which bisect the solution into a left and right portion, join in a vortex inside the subsonic area. The shock $\overleftarrow{S}_{32}$ ends at J_{34}. Below $\overrightarrow{R}_{41}$ the subsonic area is partly bounded by a shock wave.

4. Conclusions. The computational results presented here demonstrate the complex phenomenology inherent in the two-dimensional gas dynamic Riemann problem, despite the limitation that each of the four jumps consists of a single elementary wave. The predictions of [20] are close to our results in most cases where this is reasonable to expect. One discrepancy is that predicted shock waves do not appear in the computations or that we have found unpredicted shock waves. Also, for Configuration 2 and Configuration C, predicted compression waves are actually computed as shock waves over a substantial range of parameter space. These changes can have further consequences for the overall solution. However, the present work has led to the discovery of many new flowfield patterns; this is especially true for cases involving slip line initial data. For such cases we remark that some of the other cases appear as part of the flowfield pattern, as might be expected.

An important result is that many of the familiar oblique shock wave reflection (OSWR) flowfields appear here. An interesting question is whether or not OSWR is a subset of the two-dimensional Riemann problem, i.e., given a wedge angle and shock wave Mach number [5], can Riemann problem initial data be found so as to construct the given OSWR case? Note that this is nontrivial only because of the restricted set of initial data allowed.

We note here that if the four initial states were separated by two lines not necessarily perpendicular, the number of allowable distinct initial configurations would increase to 77 (respectively, 75) for isentropic (respectively, polytropic) gas. Given the many other possible generalizations, including a change of equation of state, it is easy to see that the two-dimensional Riemann problem contains many possibilities.

It would appear that this is a very negative result regarding the direct use of the two-dimensional Riemann problem in constructing an unsplit RCM (for analytical or numerical purposes) or finite difference schemes. However, the set of points in physical space-time involving such complex interactions is of high codimension, and it still might be hoped that such schemes are possible. Whether or not the present calculations will prove useful in this endeavor, we can certainly expect that the results here can be used as test or reference cases in the code development process.

Acknowledgment. The authors would like to thank Tai-Ping Liu for helpful discussions.

REFERENCES

[1] P. Colella and H. M. Glaz, *Efficient solution algorithms for the Riemann problem for real gases*, J. Comput. Phys., 59 (1985), pp. 264–289.

[2] P. Colella and P. R. Woodward, *The piecewise parabolic method* (PPM) *for gas-dynamical simulations* J. Comput. Phys., 54 (1984), pp. 174–201.

[3] R. Courant and K. O. Friedrichs, *Supersonic Flow and Shock Waves*, Springer-Verlag, New York, 1948.

[4] H. M. Glaz, *Numerical computations in gas dynamics with high resolution schemes*, in Shock Tubes and Waves, Proc. 16th Internat. Symp. Shock Tubes and Waves, H. Grönig, ed., VCH Publishers, 1988, pp. 75–88.

[5] ———, *Self-similar shock reflection in two space dimensions*, in Multidimensional Hyperbolic Problems and Computations, IMA Vol. Math. Appl., Vol. 29, J. Glimm and A. J. Majda, eds., Springer-Verlag, New York, 1991, pp. 70–88.

[6] H. M. Glaz, P. Colella, I. I. Glass, and R. L. Deschambault, *A numerical study of oblique shock-wave reflections with experimental comparisons*, Proc. Roy. Soc. London, A398 (1985), pp. 117–140.

[7] H. M. Glaz, P. A. Walter, I. I. Glass, and T. C. J. Hu, *Oblique shock wave reflections in* SF_6*: a comparison of calculation and experiment*, AIAA J. Prog. Astr. Aero., 106 (1986), pp. 359–387.

[8] J. GLIMM, *Solutions in the large for nonlinear hyperbolic systems of equations*, Comm. Pure Appl. Math., 18 (1965), pp. 697–715.

[9] J. GLIMM, C. KLINGENBERG, O. MCBRYAN, B. PLOHR, D. SHARP, AND S. YANIV, *Front tracking and two-dimensional Riemann problems*, Adv. in Appl. Math., 6 (1985), pp. 259–290.

[10] J. GUCKENHEIMER, *Shocks and rarefactions in two space dimensions*, Arch. Rational Mech. Anal., 59 (1975), pp. 281–291.

[11] W. B. LINDQUIST, *The scalar Riemann problem in two spatial dimensions: piecewise smoothness of solutions and its breakdown*, SIAM J. Math. Anal., 17 (1986), pp. 1178–1197.

[12] T.-P. LIU, *Admissible solutions of hyperbolic conservation laws*, Mem. Amer. Math. Soc., No. 240, 30 (1981).

[13] A. MAJDA, *The stability of multi-dimensional shock fronts*, Mem. Amer. Math. Soc., No. 275, 41 (1983).

[14] ———, *Compressible Fluid Flow and Systems of Conservation Laws in Several Space Variables*, Springer-Verlag, New York, 1984.

[15] C. W. SCHULZ-RINNE, *Classification of the Riemann problemfor two-dimensional gas dynamics*, SIAM J. Math. Anal., 24 (1993), pp. 76–88.

[16] J. A. SMOLLER, *Shock Waves and Reaction-Diffusion Equations*, Springer-Verlag, New York, 1982.

[17] D. H. WAGNER, *The Riemann problem in two space dimensions for a single conservation law*, SIAM J. Math. Anal., 14 (1983), pp. 534–559.

[18] T. CHANG (T. ZHANG), L. HSIAO (L. XIAO), *The Riemann Problem and Interaction of Waves in Gas Dynamics*, Longman Scientific and Technical, Pitman Monographs No. 41, Essex, 1989.

[19] T. ZHANG AND Y. ZHENG, *Two-dimensional Riemann problem for a scalar conservation law*, Trans. Amer. Math. Soc., 312 (1989), pp. 589–619.

[20] ———, *Conjecture on the structure of solutions of the Riemann problem for two-dimensional gas dynamic systems*, SIAM J. Math. Anal., 21 (1990), pp. 593–630.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1415–1436, November 1993

CONSTRUCTION OF *K*-DIMENSIONAL DELAUNAY TRIANGULATIONS USING LOCAL TRANSFORMATIONS*

BARRY JOE†

Abstract. In [SIAM J. Sci. Statist. Comput., 10 (1989), pp. 718–741] and [Comput. Aided Geom. Des., 8 (1991), pp. 123–142] the author presented algorithms that use local transformations to construct a Delaunay triangulation of a set of n three-dimensional points. This paper proves that local transformations can be used to construct a Delaunay triangulation of a set of n k-dimensional points for any $k \geq 2$, and presents algorithms using this approach. The empirical time complexities of these algorithms are discussed for sets of random points from the uniform distribution as well as worst-case time complexities. These time complexities are about the same or better than those of other algorithms for constructing k-dimensional Delaunay triangulations (when $k \geq 3$).

Key words. k-dimensional triangulation, Delaunay triangulation, Voronoi tessellation, local transformations, computational geometry, mesh generation

AMS subject classifications. 65N50, 68Q20, 68U05

1. Introduction. Given n k-dimensional (k-D) points (i.e., points in Euclidean space E^k), the k-D Delaunay triangulation problem is to connect them into nonoverlapping simplices that fill the convex hull of the points such that the empty circumhypersphere criterion is satisfied, i.e., the hypersphere passing through the $k+1$ vertices of any simplex of the triangulation contains none of the given n points in its interior. (A simplex is the convex hull of $k+1$ affinely independent points; it is a triangle in two dimensions and a tetrahedron in three dimensions.) A Delaunay triangulation is unique if no $k+2$ of the n given points are on the same hypersphere.

The Delaunay triangulation is also the dual of the Voronoi tessellation of the n given points or sites, where the Voronoi tessellation is the partitioning of E^k into n subregions such that each subregion contains the points of E^k that are closer to one site than all the other sites [22], [5]. Given the Delaunay triangulation, it is straightforward to construct the Voronoi tessellation in time proportional to the size of the triangulation (and vice versa). Two- and three-dimensional Delaunay triangulations are used for finite element mesh generation [3], [10], [11], [14], [25]. k-D Delaunay triangulations are used for surface interpolation and contouring [17], [18], [20]. Other applications of the two geometric structures are given in [2] and [29].

We are interested in algorithms that can construct k-D Delaunay triangulations (or Voronoi tessellations) for any $k \geq 2$. Some algorithms work only in two dimensions (so far), e.g., the $O(n \log n)$ divide and conquer approach [22], [19] and sweepline approach [9]. The edge swapping approach can be used for constructing two-dimensional Delaunay triangulations [17], [28]. This local transformation approach was recently extended to the three-dimensional case [12], [13], where face swaps are used instead of edge swaps. Lawson has shown that there are two different ways to triangulate some sets of $k+2$ k-D points [18]. Hence local transformations are also possible in k dimensions, $k \geq 4$. In this paper, we prove that local transformations can be used to construct k-D Delaunay triangulations using an incremental approach, and we present algorithms that are k-D versions of those in [13].

*Received by the editors October 23, 1991; accepted for publication (in revised form) December 2, 1992. This work was partially supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

†Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1 (barry@cs.ualberta.ca).

Other algorithms for constructing k-D Delaunay triangulations (or Voronoi tessellations) are given in [29], [2], [1], and [5]. Watson's algorithm [29] uses an incremental approach in which the Delaunay triangulation of the first i data points is updated from the Delaunay triangulation of the first $i-1$ data points by first deleting all simplices whose circumhypershere contains the ith data point and then adding the simplices involving the ith data point. Bowyer's algorithm [2] also uses an incremental approach, but updates the Voronoi tessellation instead of the Delaunay triangulation. The estimated time complexities of these two algorithms (for sets of random points) are $O(n^{(2k-1)/k})$ and $O(n^{1+1/k})$, respectively. The worst-case time complexities are not discussed in these two papers. We are assuming here that k is a small fixed constant; as a function of k and n, the time complexities would also include a factor involving k, e.g., k^3, which depends on implementation details.

The algorithm by Avis and Bhattacharya [1] constructs a Delaunay triangulation by finding one simplex at a time; from an unmatched facet of an existing simplex, a search is made for the vertex of the other simplex sharing this facet. For sets of random points from the uniform distribution, the empirical time complexity of this algorithm is $O(n^2)$ for any k. The worst-case time complexity is $O(n^{\lceil k/2\rceil+1})$. This is because in the worst-case, the number of simplices in a Delaunay triangulation is $O(n^{\lceil k/2\rceil})$ [5].

An algorithm for constructing k-D Delaunay triangulations based on their relationship with $(k+1)$-D convex hulls is described in [5]. This algorithm projects the n k-D data points onto a paraboloid in $k+1$ dimensions and constructs the convex hull of the $(k+1)$-D points; the k-D Delaunay triangulation is then obtained from an appropriate portion of the convex hull. The worst-case time complexity of this algorithm is $O(n \log n)$ for $k = 2$ and $O(n^{\lfloor k/2\rfloor+1})$ for $k \geq 3$; these complexities are based on the worst-case time complexities of algorithms for constructing convex hulls [21], [26]. Since $\lceil k/2\rceil = \lfloor k/2\rfloor + 1$ for odd k, the time complexity is worst-case optimal for odd k. The time complexity for sets of random points is not discussed in [5].

Of the three algorithms based on local transformations presented in this paper, the first has an empirical time complexity of $O(n(\log n)^{k-1})$ for sets of random points from the uniform distribution, and a worst-case time complexity of $O(n^{\lfloor k/2\rfloor+1})$. Thus this first algorithm is also worst-case optimal for odd k. The empirical and worst-case time complexities of the other two algorithms are theoretically not as good, but in practice they are faster than the first algorithm. Our algorithms use an incremental approach as in Watson's algorithm (i.e., the Delaunay triangulation of the first i data points is updated from the Delaunay triangulation of the first $i-1$ data points), but using local transformations, the deletion and addition of simplices are intermixed when adding the ith data point.

This paper is organized as follows. In §2, we describe local transformations in k dimensions. In §3, we describe an incremental approach for constructing k-D Delaunay triangulations using local transformations. In §4, we prove that this approach always constructs a Delaunay triangulation. In §5, we describe three algorithms and a data structure based on this approach. In §6, we discuss the time complexities of the algorithms and present experimental results from our implementation of these algorithms.

2. Local transformations. We first give some terminology before describing local transformations. In k dimensions, a (nondegenerate) *simplex* is the convex hull of $k+1$ affinely independent points, a (nondegenerate) *j-face* is the convex hull of $j+1$ affinely independent points, and a *facet* is a $(k-1)$-face. A degenerate simplex or degenerate j-face is the convex hull of $k+1$ or $j+1$ affinely dependent points, respectively. In this paper, simplices and faces are nondegenerate unless preceded by the word "degenerate."

Let $\mathcal{V}$ be a set of $n \geq k+1$ distinct k-D points, not all in the same hyperplane, and let $\mathcal{C}$ be the convex hull of $\mathcal{V}$. A *triangulation* of $\mathcal{V}$ is a set $\mathcal{S}$ of simplices satisfying the following properties:

(a) All vertices of each simplex of $\mathcal{S}$ are members of $\mathcal{V}$.

(b) Each simplex of $\mathcal{S}$ contains no points of $\mathcal{V}$ other than its vertices.

(c) The union of the simplices of $\mathcal{S}$ is $\mathcal{C}$.

(d) The interiors of the simplices of $\mathcal{S}$ are pairwise disjoint.

(e) Each facet of a simplex of $\mathcal{S}$ is either on the boundary of $\mathcal{C}$ (called a *boundary facet*) or is common to exactly two simplices of $\mathcal{S}$ (called an *interior facet*).

Let $a_1, a_2, \ldots, a_{k+2}$ be $k+2$ distinct k-D points, not all in the same hyperplane. Then there are either one or two ways to triangulate these $k+2$ points, and there are a finite number of different configurations of the $k+2$ points such that the configuration type determines how many simplices are present in the one or two possible triangulations. These results are proved in [18]. If two different triangulations are possible, then a *local transformation* is defined to be the replacement of one of these triangulations by the other.

For example, in three dimensions, the five different configurations are shown in Fig. 1. In the first two configurations, no four of the five points are coplanar; line segment a_4a_5 intersects the interior of facet $a_1a_2a_3$ in configuration 1 and a_1 is in the interior of the convex hull in configuration 2. In the last three configurations, a_1, a_2, a_4, and a_5 are coplanar and form the possible configurations of four planar points, so these can be considered to be degenerate three-dimensional configurations. Local transformations are possible only if the five points are in configuration 1 or 3; the triangulation changes from the A to the B version or vice versa. In configuration 1, the interior facet $a_1a_2a_3$ in triangulation A is swapped for the interior facets $a_1a_4a_5$, $a_2a_4a_5$, $a_3a_4a_5$ in triangulation B. In configuration 3, the edge a_1a_2 in triangulation A is swapped for the edge a_4a_5 in triangulation B (this is the diagonal edge swap used for two-dimensional triangulations).

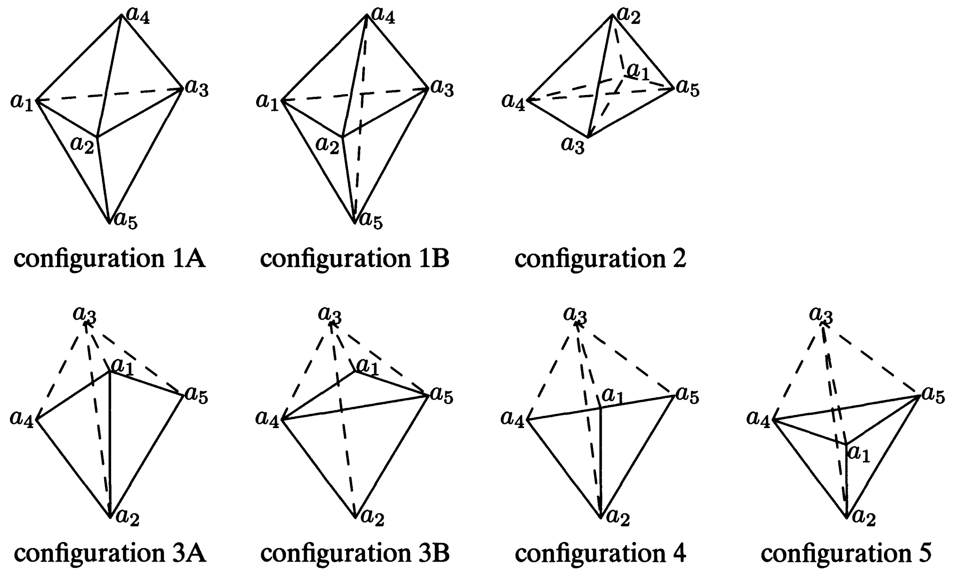


FIG. 1. *Tetrahedra are* (1A) $a_1a_2a_3a_4$, $a_1a_2a_3a_5$; (1B) $a_1a_2a_4a_5$, $a_1a_3a_4a_5$, $a_2a_3a_4a_5$; (2) $a_1a_2a_3a_4$, $a_1a_2a_3a_5$, $a_1a_2a_4a_5$, $a_1a_3a_4a_5$; (3A) $a_1a_2a_3a_4$, $a_1a_2a_3a_5$; (3B) $a_1a_3a_4a_5$, $a_2a_3a_4a_5$; (4) $a_1a_2a_3a_4$, $a_1a_2a_3a_5$; (5) $a_1a_2a_3a_4$, $a_1a_2a_3a_5$, $a_1a_3a_4a_5$.

In k dimensions, the type of configuration and the number of simplices in the one or two possible triangulations can be determined as follows [18]. From the $k+2$ vertices $a_1, \ldots, a_{k+2}$, let S_i denote the (possibly degenerate) simplex formed from $k+1$ of these vertices with a_i excluded, and let $F_{i,j}$ denote the (possibly degenerate) facet formed from k of these vertices with a_i and a_j excluded. Suppose the vertices are labeled so that S_{k+1} and S_{k+2} are nondegenerate simplices sharing common facet $F_{k+1,k+2}$ with a_{k+1} and a_{k+2} on opposite sides of the hyperplane containing $F_{k+1,k+2}$ (this labeling is always possible). Let $(\alpha_1, \alpha_2, \ldots, \alpha_{k+1})$ be the barycentric coordinates of a_{k+2} with respect to $a_1, \ldots, a_{k+1}$, i.e., $\sum_{i=1}^{k+1} \alpha_i a_i = a_{k+2}$ and $\sum_{i=1}^{k+1} \alpha_i = 1$, and let $\alpha_{k+2} = -1$. Note that the labeling of vertices implies that $\alpha_{k+1} < 0$. For $1 \le i \le k+2$, define i (a_i) to be a negative, positive, or zero index (vertex) if α_i is negative, positive, or zero, respectively. If z is a zero index, then the $k+1$ vertices a_i, $i \ne z$, lie on the same hyperplane. Let σ_N, σ_P, and σ_0 be the number of negative, positive, and zero indices, respectively. Note that $\sigma_N \ge 2$, $\sigma_P \ge 1$, and $\sigma_N + \sigma_P + \sigma_0 = k+2$.

The type of configuration determined by $a_1, \ldots, a_{k+2}$ is characterized by $\sigma = (\sigma_N, \sigma_P, \sigma_0)$. (Two sets of $k+2$ vertices have the same type of configuration if and only if their σ vectors are identical or differ only by the interchange of the σ_N and σ_P values. Any configuration with $\sigma_N \ge 2$, $\sigma_P \ge 1$, $\sigma_0 \ge 0$, and $\sigma_N + \sigma_P + \sigma_0 = k+2$ is possible.) Let $\mathcal{S}_N$ and $\mathcal{S}_P$ be the sets of simplices S_i with negative and positive indices, respectively (if i is a zero index, then S_i is degenerate). If $\sigma_P = 1$, then only one triangulation of the $k+2$ vertices is possible, and it contains the σ_N simplices of $\mathcal{S}_N$; the simplex S_j, where j is the positive index, is nondegenerate but is not a valid simplex of a triangulation because it contains the point a_j. If $\sigma_P \ge 2$, then exactly two triangulations of the $k+2$ vertices are possible; the first contains the σ_N simplices of $\mathcal{S}_N$, and the second contains the σ_P simplices of $\mathcal{S}_P$. Each simplex of $\mathcal{S}_N$ ($\mathcal{S}_P$) is adjacent to (shares an interior facet with) all the other simplices of $\mathcal{S}_N$ ($\mathcal{S}_P$ if $\sigma_P \ge 2$).

We have the $\sigma_P \ge 2$ case when a local transformation is possible. If $\sigma_0 \ge 1$, the local transformation can be considered to be applied in $k - \sigma_0$ dimensions as follows. For a negative or positive index i, let E_i be the $(k - \sigma_0)$ face consisting of the negative and positive vertices, excluding a_i. Let $\mathcal{E}_N$ ($\mathcal{E}_P$) be the set of σ_N (σ_P) faces E_i with negative (positive) index i. The local transformation swaps the faces of $\mathcal{E}_N$ for the faces of $\mathcal{E}_P$, with the σ_0 zero vertices being part of all simplices.

Since our Delaunay triangulation algorithms are based on processing and swapping facets, we now describe the facets involved in a local transformation from the simplices of triangulation $\mathcal{S}_N$ to the simplices of triangulation $\mathcal{S}_P$. Let $\mathcal{F}_N$, $\mathcal{F}_P$, $\mathcal{F}_B$, $\mathcal{F}_{BN}$, and $\mathcal{F}_{BP}$ be the sets of interior facets of $\mathcal{S}_N$, interior facets of $\mathcal{S}_P$, boundary facets common to both $\mathcal{S}_N$ and $\mathcal{S}_P$, boundary facets unique to $\mathcal{S}_N$, and boundary facets unique to $\mathcal{S}_P$, respectively. $\mathcal{F}_N$ contains the $\sigma_N(\sigma_N - 1)/2$ facets $F_{i,j}$ where i and j are negative indices. $\mathcal{F}_P$ contains the $\sigma_P(\sigma_P - 1)/2$ facets $F_{i,j}$ where i and j are positive indices. $\mathcal{F}_B$ contains the $\sigma_N \sigma_P$ facets $F_{i,j}$, where i is a negative index and j is a positive index. $\mathcal{F}_{BN}$ and $\mathcal{F}_{BP}$ are nonempty if and only if $\sigma_0 \ge 1$. $\mathcal{F}_{BN}$ contains the $\sigma_N \sigma_0$ facets $F_{i,j}$, where i is a negative index and j is a zero index. $\mathcal{F}_{BP}$ contains the $\sigma_P \sigma_0$ facets $F_{i,j}$, where i is a positive index and j is a zero index. If $\sigma_0 \ge 2$, then facet $F_{i,j}$ with zero indices i, j is degenerate.

3. Delaunay triangulations from local transformations.

3. Delaunay triangulations from local transformations. In this section, we describe an incremental approach for constructing k-D Delaunay triangulations using local transformations. Let $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ be a set of $n \ge k+1$ distinct k-D vertices, not all in the same hyperplane. A *Delaunay triangulation* of $\mathcal{V}$ is a triangulation that satisfies the (global) empty circumhypersphere criterion, i.e., the hypersphere passing through the $k + 1$ vertices of any simplex of the triangulation contains none of the n vertices in its interior. A Delaunay triangulation also satisfies a local empty circumhypersphere criterion as indicated by the following definition and theorem from [18].

DEFINITION 1. Let $\mathbf{a} = a_1a_2 \dots a_k$ be an interior facet of a triangulation of $\mathcal{V}$, and let $\mathbf{a}b$ and $\mathbf{a}c$ be the simplices sharing facet $\mathbf{a}$ (with vertices b, c on opposite sides of the hyperplane containing $\mathbf{a}$). Then $\mathbf{a}$ is said to be *locally optimal* if and only if the circumhypersphere of $\mathbf{a}b$ does not contain c in its interior.

THEOREM 1 [18]. *A triangulation $\mathcal{T}$ is a Delaunay triangulation if and only if every interior facet of $\mathcal{T}$ is locally optimal.*

Note that Definition 1 is not affected by the labeling of b, c, and Theorem 1 provides a simple way to check whether or not a given triangulation $\mathcal{T}$ is Delaunay. This definition and theorem lead to the following corollary. A different version of part of this corollary is given and proved in [18].

COROLLARY 1. *Let $a_1, a_2, \dots, a_{k+2}$ be $k + 2$ distinct k-D vertices, not all in the same hyperplane. If $\mathcal{T}$ is the unique triangulation of these vertices, then all interior facets of $\mathcal{T}$ are locally optimal. If there are two different triangulations $\mathcal{T}$, $\mathcal{T}'$ of these vertices, then either all interior facets of $\mathcal{T}$ ($\mathcal{T}'$) are locally optimal or all interior facets of $\mathcal{T}$ ($\mathcal{T}'$) are not locally optimal, and the latter case cannot occur for both $\mathcal{T}$ and $\mathcal{T}'$.*

Proof. If $\mathcal{T}$ is the unique triangulation, then it must be a Delaunay triangulation (since this always exists), so all interior facets of $\mathcal{T}$ are locally optimal by Theorem 1. Suppose two triangulations are possible. If $a_1, a_2, \dots, a_{k+2}$ lie on the same hypersphere, then both triangulations are Delaunay, otherwise exactly one of the triangulations is Delaunay. Suppose $\mathcal{T}$ is not Delaunay. By Theorem 1, there exists an interior facet F of $\mathcal{T}$ that is not locally optimal. Suppose F does not contain the two vertices a_i, a_j. Let S be the simplex formed from F and a_i. Then the circumhypersphere of S contains a_j in its interior. This implies that all interior facets of S are not locally optimal (since a_j is always the other vertex). Since S is adjacent to all other simplices of $\mathcal{T}$, it follows that the circumhypersphere of each simplex of $\mathcal{T}$ contains the other vertex (not involved in the simplex) in its interior, and all interior facets of $\mathcal{T}$ are not locally optimal. □

Theorem 1 and Corollary 1 suggest that one approach for constructing a Delaunay triangulation of $\mathcal{V}$ is to apply local transformations to nonlocally optimal interior facets to remove them from a triangulation of $\mathcal{V}$. The following definitions suggest that removing these facets may not be straightforward.

DEFINITION 2. Let $\mathbf{a} = a_1a_2 \dots a_k$ be an interior facet of a triangulation $\mathcal{T}$ of $\mathcal{V}$, and let $\mathbf{a}a_{k+1}$ and $\mathbf{a}a_{k+2}$ be the simplices of $\mathcal{T}$ sharing facet $\mathbf{a}$. Suppose the notation of §2 is used. If $\sigma_0 = 0$, then $\mathbf{a}$ is said to be *transformable* if and only if $\sigma_P \geq 2$ and the σ_N simplices of $\mathcal{S}_N$ are all in $\mathcal{T}$. If $\sigma_0 \geq 1$, then $\mathbf{a}$ is said to be *transformable* if and only if (1) $\sigma_P \geq 2$, (2) the σ_N $(k - \sigma_0)$-faces of $\mathcal{E}_N$ are all in $\mathcal{T}$, and (3) for any two faces E_i, E_j of $\mathcal{E}_N$, $E_ib_1 \dots b_{\sigma_0}$ (the simplex containing face E_i and vertices $b_1, \dots, b_{\sigma_0}$) is a simplex of $\mathcal{T}$ if and only if $E_jb_1 \dots b_{\sigma_0}$ is a simplex of $\mathcal{T}$.

If $\mathbf{a}$ is nontransformable, then a local transformation cannot be applied to remove $\mathbf{a}$ from the triangulation. If $k = 2$, then any nonlocally optimal interior facet (edge) a_1a_2 is transformable since, by Corollary 1, $\sigma_N = \sigma_P = 2$ and $\sigma_0 = 0$ for this facet. Suppose $k \geq 3$ and $\sigma_0 \geq 1$. If interior facet $\mathbf{a}$ is transformable, then $q \geq 1$ simultaneous local transformations can be performed to remove $\mathbf{a}$, where q is the number of simplices of $\mathcal{T}$ incident on a face E_i of $\mathcal{E}_N$. Note that these local transformations must be performed simultaneously in order to guarantee that a valid triangulation is produced.

For example, suppose $k = 3$, $\sigma_0 = 1$, and the tetrahedra are $a_1a_2a_3a_4$ and $a_1a_2a_3a_5$ as in configuration 3A of Fig. 1, as well as $a_1a_2a_6a_4$ and $a_1a_2a_6a_5$ (with a_6 and a_3 on opposite sides of $a_1a_2a_4$). Then two simultaneous local transformations can be performed to produce tetrahedra $a_4a_5a_3a_1$ and $a_4a_5a_3a_2$ as in configuration 3B of Fig. 1, as well as $a_4a_5a_6a_1$ and $a_4a_5a_6a_2$.

The following examples illustrate how nontransformable facets can arise in a triangulation $\mathcal{T}$.

(a) $k = 3$, $\sigma_0 = 0$: Suppose vertices $a_1, \ldots, a_5$ are as in configuration 1B of Fig. 1 with tetrahedra $a_1a_4a_5a_2$ and $a_1a_4a_5a_3$, but tetrahedron $a_2a_3a_4a_5$ is replaced with tetrahedra $a_2a_4a_5a_6$ and $a_3a_4a_5a_6$ (where a_6 lies between a_2 and a_3 in circular order around a_4a_5). Then $a_1a_4a_5$ is nontransformable.

(b) $k = 3$, $\sigma_0 = 1$: Suppose the tetrahedra are $a_1a_2a_3a_4$ and $a_1a_2a_3a_5$ as in configuration 3A of Fig. 1, $a_1a_2a_4$ and $a_1a_2a_5$ are interior facets of $\mathcal{T}$, and the other two tetrahedra incident on these two facets are $a_1a_2a_4b$ and $a_1a_2a_5c$ with $b \neq c$. Then $a_1a_2a_3$ is nontransformable.

(c) $k = 4$, $\sigma_0 = 1$: Suppose the simplices are $a_1a_4a_5a_6a_2$ and $a_1a_4a_5a_6a_3$, where $a_1, \ldots, a_5$ are in the same hyperplane as in configuration 1B of Fig. 1. Then $a_1a_4a_5a_6$ is nontransformable if either facet $a_2a_3a_4a_5$ is not in $\mathcal{T}$ or simplex $a_2a_3a_4a_5a_6$ is not in $\mathcal{T}$ or $a_1a_2a_4a_5b$, $a_1a_3a_4a_5c$, $a_2a_3a_4a_5d$ are simplices of $\mathcal{T}$ such that b, c, and d are not all the same vertex.

(d) $k = 4$, $\sigma_0 = 2$: Suppose the simplices are $a_1a_2a_3a_6a_4$ and $a_1a_2a_3a_6a_5$ where a_1, a_2, a_4, a_5 are in the same hyperplane as in configuration 3A of Fig. 1. Let $\mathcal{E}_4$ ($\mathcal{E}_5$) be the set of edges b_1b_2 such that $a_1a_2a_4b_1b_2$ ($a_1a_2a_5b_1b_2$) is a simplex of $\mathcal{T}$. Then $a_1a_2a_3a_6$ is nontransformable if the sets $\mathcal{E}_4$ and $\mathcal{E}_5$ are not identical.

DEFINITION 3. A triangulation $\mathcal{T}$ of $\mathcal{V}$ is said to be *pseudolocally optimal* if every nonlocally optimal interior facet of $\mathcal{T}$ is nontransformable.

Any triangulation of $\mathcal{V}$ can be transformed to a pseudolocally optimal triangulation by a finite sequence of local transformations applied to nonlocally optimal transformable interior facets, i.e., the sequence of triangulations cannot cycle. This is proved in [12] for the case $k = 3$, and the proof can be easily extended to any dimension k. For $k = 2$, a pseudolocally optimal triangulation is always Delaunay since nonlocally optimal nontransformable facets (edges) cannot occur. However, for $k \geq 3$, a pseudolocally optimal triangulation may not be Delaunay. An example of a pseudolocally optimal non-Delaunay triangulation for $k = 3$ is given in [12]. By changing the three-dimensional points (x_i, y_i, z_i) in this example to k-D points $(x_i, y_i, z_i, 0, \ldots, 0)$ and adding the $k-3$ k-D points $(0,0,0,1,0,\ldots,0), \ldots, (0,0,0,0,\ldots,0,1)$ to get k-D simplices from the tetrahedra, an example of a pseudolocally optimal non-Delaunay k-D triangulation for any $k \geq 4$ is obtained.

The existence of pseudolocally optimal non-Delaunay triangulations implies that a Delaunay triangulation cannot generally be constructed by applying local transformations in any order to any triangulation. To guarantee that a Delaunay triangulation of $\mathcal{V}$ is constructed, we use the following incremental approach. Suppose the vertices of $\mathcal{V}$ are relabeled if necessary so that $v_1, \ldots, v_{i-1}$ are not in the same hyperplane and a Delaunay triangulation $\mathcal{T}_{i-1}^D$ of $v_1, \ldots, v_{i-1}$ has been constructed (when $i = k+2$, the Delaunay triangulation consists of a single simplex). If $i \leq n$, then vertex v_i is inserted into $\mathcal{T}_{i-1}^D$ and an initial triangulation $\mathcal{T}_i^{(0)}$ involving v_i is constructed as follows.

If v_i lies outside the convex hull of $v_1, \ldots, v_{i-1}$, then simplex $\mathbf{a}v_i$ is added to $\mathcal{T}_{i-1}^D$ for every boundary facet $\mathbf{a}$ of $\mathcal{T}_{i-1}^D$ that is visible from v_i ($\mathbf{a}$ is visible from v_i if v_i and any point w in the interior of the convex hull of $v_1, \ldots, v_{i-1}$ are on opposite sides of the hyperplane containing $\mathbf{a}$). If v_i lies in the interior of simplex $S = a_1a_2 \ldots a_{k+1}$ of $\mathcal{T}_{i-1}^D$, then S is deleted and replaced with the $k+1$ simplices S_j, where S_j is S with vertex a_j

replaced by v_i, $j = 1, \dots, k+1$. We have the remaining case when v_i lies in the interior of $(p-1)$-face $\mathbf{a} = a_1 a_2 \dots a_p$ of $\mathcal{T}_{i-1}^D$, where $2 \le p \le k$. In this case, any simplex $S = \mathbf{a} b_1 \dots b_{k+1-p}$ of $\mathcal{T}_{i-1}^D$ containing face $\mathbf{a}$ is deleted and replaced with the p simplices S_j, where S_j is S with vertex a_j replaced by v_i, $j = 1, \dots, p$.

In the next section, we prove that $\mathcal{T}_i^{(0)}$ can be transformed into a Delaunay triangulation $\mathcal{T}_i^D$ by a finite sequence of local transformations applied to nonlocally optimal transformable interior facets.

4. Proof of Delaunay triangulation construction. The proof that $\mathcal{T}_i^{(0)}$ can be transformed into a Delaunay triangulation $\mathcal{T}_i^D$ is just a generalization of the proof for the three-dimensional case given in [13]. To simplify the proofs of lemmas, we assume that no $k+1$ points of $\mathcal{V}$ lie on the same hyperplane and that no $k+2$ points of $\mathcal{V}$ lie on the same hypersphere. The former assumption implies that all local transformations are nondegenerate, i.e., $\sigma_0 = 0$. The latter assumption is only needed for the last lemma of this section. These assumptions apply to the proofs only; the lemmas still hold in the degenerate cases (this is indicated by parenthesized phrases). The extension of the proofs to handle the degenerate cases can be done in a way similar to that in [13] for $k = 3$, where the two assumptions are not made.

First, we give some notation. The interior of the circumhypersphere of simplex $a_1 a_2 \dots a_{k+1}$ is denoted by $\bigcirc a_1 a_2 \dots a_{k+1}$. For $r \ge 1$, let $\mathcal{T}_i^{(r)}$ be the triangulation obtained by applying a local transformation to a nonlocally optimal transformable interior facet of $\mathcal{T}_i^{(r-1)}$ (or $q \ge 1$ simultaneous local transformations if $\sigma_0 \ge 1$). For $r \ge 0$, let $\mathcal{F}_i^{(r)} = \{\text{facets } \; a_1 \dots a_k \mid a_1 \dots a_k v_i \text{ is a simplex in } \mathcal{T}_i^{(r)}\}$ and $\mathcal{G}_i^{(r)} = \{\text{facets } \; a_1 \dots a_{k-1} v_i \mid \text{there exists } a_k \text{ such that } a_1 \dots a_{k-1} a_k v_i \text{ is a simplex in } \mathcal{T}_i^{(r)}\}$. If $\mathbf{a} v_i = a_1 \dots a_{k-1} v_i$ is an interior facet of $\mathcal{G}_i^{(r)}$ incident on simplices $\mathbf{a} b v_i$ and $\mathbf{a} c v_i$, then let $\theta_i^{(r)}(\mathbf{a})$ denote the angle at $\mathbf{a}$ between (the hyperplanes containing) facets $\mathbf{a} b$ and $\mathbf{a} c$ as viewed from v_i, i.e., the angle is taken to be larger than π if and only if c and v_i are on opposite sides of the hyperplane containing $\mathbf{a} b$.

Since $\mathcal{T}_{i-1}^D$ is Delaunay, all interior facets in $\mathcal{T}_{i-1}^D$ are locally optimal by Theorem 1. This implies that in $\mathcal{T}_i^{(0)}$, all nonlocally optimal interior facets are in $\mathcal{F}_i^{(0)} \cup \mathcal{G}_i^{(0)}$ (since the two simplices incident on other interior facets are unchanged), and for all interior facets $\mathbf{a} v_i$ in $\mathcal{G}_i^{(0)}$, $\theta_i^{(0)}(\mathbf{a}) \ge \pi$ $(< \pi)$ if v_i is outside the convex hull of $v_1, \dots, v_{i-1}$ (in the interior of a simplex of $\mathcal{T}_{i-1}^D$). To show that $\mathcal{T}_i^{(m)}$ is Delaunay for some m, we first show that the local transformations applied to the $\mathcal{T}_i^{(r)}$ are type-N transformations, where a *type-N transformation* is a local transformation in which v_i is one of the $k+2$ vertices and is a negative vertex (using the notation of §2). In other words, a type-N transformation replaces σ_N simplices, exactly one of which does not have v_i as a vertex, by σ_P simplices, all having v_i as a vertex. For $k = 3$, Fig. 2 illustrates the two different kinds of nondegenerate type-N transformations.

LEMMA 1. *Suppose for all $r < m$, the local transformation applied to $\mathcal{T}_i^{(r)}$ is a type-N transformation (or simultaneous local transformations are type-N transformations). For all $r \le m$:*

(A) *all nonlocally optimal interior facets of $\mathcal{T}_i^{(r)}$ are in $\mathcal{F}_i^{(r)} \cup \mathcal{G}_i^{(r)}$, and*

(B) *if $a_1 a_2 \dots a_{k+1}$ is a simplex in $\mathcal{T}_i^{(r)}$ such that none of its vertices is v_i, then $\bigcirc a_1 a_2 \dots a_{k+1}$ does not contain any vertex except possibly vertex v_i.*

Proof (by induction). Statements (A) and (B) are clearly true for $r = 0$. Suppose they are true for $0 \le r < m$ and $\mathcal{T}_i^{(r)}$ is not pseudolocally optimal. Suppose that a

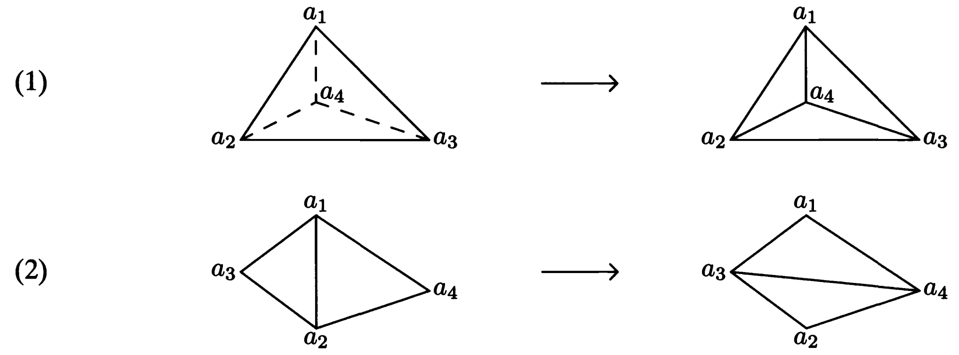


FIG. 2. *Two kinds of nondegenerate type-N transformations for* $k = 3$ (*"before" on left, "after" on right*). *Vertex* v_i *is not shown; imagine that it is above the page. Facets in* $\mathcal{F}_i^{(r)}$ *are solid lines, other edges are dashed lines.* (1) a_4 *below* $a_1a_2a_3$, a_4v_i *intersects* $a_1a_2a_3$: $a_1a_2a_3v_i$, $a_1a_2a_3a_4 \rightarrow a_1a_2a_4v_i$, $a_1a_3a_4v_i$, $a_2a_3a_4v_i$. (2) $\theta_i^{(r)}(a_1a_2) > \pi$, a_1a_2 *intersects* $a_3a_4v_i$: $a_1a_2a_3v_i$, $a_1a_2a_4v_i$, $a_1a_2a_3a_4 \rightarrow a_1a_3a_4v_i$, $a_2a_3a_4v_i$.

type-N transformation is applied to nonlocally optimal transformable facet $a_1 \ldots a_k$ of $\mathcal{T}_i^{(r)}$ to get $\mathcal{T}_i^{(r+1)}$. Let the other two vertices of the type-N transformation be a_{k+1} and $a_{k+2} = v_i$. Using the notation of §2,

$$\mathcal{F}_i^{(r+1)} = (\mathcal{F}_i^{(r)} - \{F_{j,k+2} \mid j \text{ is a negative index}\}) \cup \{F_{j,k+2} \mid j \text{ is a positive index}\},$$

$$\mathcal{G}_i^{(r+1)} = (\mathcal{G}_i^{(r)} - \{F_{j,l} \mid j,l \text{ are negative indices } < k+2\}) \cup \{F_{j,l} \mid j,l \text{ are positive indices}\}.$$

(Note that the deleted facets are in $\mathcal{F}_N$.) The new interior facets of $\mathcal{T}_i^{(r+1)}$ that may become nonlocally optimal are in

$$\mathcal{F}_B = \{F_{j,l} \mid j \text{ is a positive index and } l \text{ is a negative index}\},$$

and these facets are all in $\mathcal{F}_i^{(r+1)} \cup \mathcal{G}_i^{(r+1)}$. There are no new simplices in $\mathcal{T}_i^{(r+1)}$ not involving v_i. Hence (A) and (B) are true for $r + 1$. □

LEMMA 2. *Suppose for all* $r < m$, *the local transformation applied to* $\mathcal{T}_i^{(r)}$ *is a type-N transformation* (*or simultaneous local transformations are type-N transformations*). *For all* $r \leq m$:

(C) *for all interior facets* $a_1 \ldots a_{k-1}v_i$ *in* $\mathcal{G}_i^{(r)}$, *if* $\theta_i^{(r)}(a_1 \ldots a_{k-1}) \leq \pi$, *then facet* $a_1 \ldots a_{k-1}v_i$ *is locally optimal.*

Proof (by induction). With the assumption that no $k + 1$ points of $\mathcal{V}$ lie on the same hyperplane, an interior facet $\mathbf{a}v_i = a_1 \ldots a_{k-1}v_i$ in $\mathcal{G}_i^{(0)}$ satisfies $\theta_i^{(0)}(\mathbf{a}) > \pi$ if v_i is outside the convex hull of $v_1, \ldots, v_{i-1}$, and $\theta_i^{(0)}(\mathbf{a}) < \pi$ if v_i is in the interior of a simplex S of $\mathcal{T}_{i-1}^D$. In the latter case, $\mathbf{a}v_i$ is locally optimal by Corollary 1 since the triangulation of v_i and the vertices of S are unique. Hence statement (C) is true for $r = 0$.

Suppose (C) is true for $0 \leq r < m$. Suppose that a type-N transformation is applied to nonlocally optimal transformable facet $a_1 \ldots a_k$ of $\mathcal{T}_i^{(r)}$ to get $\mathcal{T}_i^{(r+1)}$. Let the other two vertices of the type-N transformation be a_{k+1} and $a_{k+2} = v_i$. Using the notation of §2, $\mathcal{F}_P = \{F_{j,l} \mid j,l \text{ are positive indices}\}$ contains the only facets of $\mathcal{G}_i^{(r+1)}$ not in $\mathcal{G}_i^{(r)}$. Suppose $F_{j,l} \in \mathcal{F}_P$ and $\mathbf{a}$ is the $(k-2)$-face of $F_{j,l}$ that does not contain v_i. Then $\theta_i^{(r+1)}(\mathbf{a}) < \pi$ since $\mathbf{a}a_j$ and $\mathbf{a}a_l$ are in $\mathcal{F}_B$, and $F_{j,l}$ is locally optimal by Corollary 1 since $a_1 \ldots a_k$ is nonlocally optimal.

The only facets of $\mathcal{G}_i^{(r+1)} \cap \mathcal{G}_i^{(r)}$ such that their $\theta_i^{(r)}$ and $\theta_i^{(r+1)}$ values are different (if they are interior facets) are in

$$\mathcal{G} = \{F_{j,l} \mid j \text{ is a positive index and } l \text{ is a negative index } < k+2\} \subset \mathcal{F}_B.$$

Suppose $F_{j,l} \in \mathcal{G}$ is an interior facet of $\mathcal{G}_i^{(r+1)}$, $\mathbf{a}$ is the $(k-2)$-face of $F_{j,l}$ that does not contain v_i, and $\theta_i^{(r+1)}(\mathbf{a}) < \pi$. Then $\theta_i^{(r)}(\mathbf{a}) < \theta_i^{(r+1)}(\mathbf{a})$ since $\mathbf{a}a_j \in \mathcal{F}_N \cap \mathcal{F}_i^{(r)}$ ($\mathbf{a}a_jv_i$ is a simplex of $\mathcal{T}_i^{(r)}$) and $\mathbf{a}a_l \in \mathcal{F}_B \cap \mathcal{F}_i^{(r+1)}$ ($\mathbf{a}a_lv_i$ is a simplex of $\mathcal{T}_i^{(r+1)}$). Let the other simplex of $\mathcal{T}_i^{(r)}$ and $\mathcal{T}_i^{(r+1)}$ incident on $F_{j,l}$ be $\mathbf{a}v_ib$. Then a_j and b are on the same side of (the hyperplane containing) $\mathbf{a}a_l$ as v_i. Since $\mathbf{a}a_ja_l$ is a simplex of $\mathcal{T}_i^{(r)}$, by Lemma 1, $\bigcirc\mathbf{a}a_ja_l$ does not contain b. The fact that $\mathbf{a}a_j$ is not locally optimal in $\mathcal{T}_i^{(r)}$ implies $\bigcirc\mathbf{a}a_ja_l$ contains v_i. Hence the part of $\bigcirc\mathbf{a}a_lv_i$ on the same side of $\mathbf{a}a_l$ as v_i is a subset of the corresponding part of $\bigcirc\mathbf{a}a_la_j$. This implies that $\bigcirc\mathbf{a}a_lv_i$ does not contain b, i.e., $\mathbf{a}v_i = F_{j,l}$ is locally optimal in $\mathcal{T}_i^{(r+1)}$. □

LEMMA 3. *For all* $m \geq 0$:

(D) *a type-N transformation* (*or simultaneous type-N transformations*) *is the only type of local transformation that can be applied to a nonlocally optimal transformable interior facet of* $\mathcal{T}_i^{(m)}$.

Proof (by induction). The basis and inductive steps are the same. Suppose $m = 0$ or statement (D) is true for all $r < m$. By Lemma 1, all nonlocally optimal interior facets of $\mathcal{T}_i^{(m)}$ are in $\mathcal{F}_i^{(m)} \cup \mathcal{G}_i^{(m)}$. First suppose that $\mathbf{a} = a_1 \dots a_k$ is a nonlocally optimal interior facet in $\mathcal{F}_i^{(m)}$ incident on simplices $\mathbf{a}a_{k+1}$ and $\mathbf{a}v_i$. Then a local transformation involving these two simplices must have v_i as a negative vertex, i.e., it must be a type-N transformation.

Now suppose that $\mathbf{a}v_i = a_1 \dots a_{k-1}v_i$ is a nonlocally optimal interior facet in $\mathcal{G}_i^{(m)}$ incident on simplices $\mathbf{a}v_ia_k$ and $\mathbf{a}v_ia_{k+1}$. By Lemma 2, $\theta_i^{(m)}(\mathbf{a}) > \pi$. This angle implies that $\mathbf{a}a_ka_{k+1}$ is a simplex in the triangulation of $a_1, \dots, a_{k+1}, v_i$ that includes $\mathbf{a}v_ia_k$ and $\mathbf{a}v_ia_{k+1}$. By Corollary 1, there is another triangulation of $a_1, \dots, a_{k+1}, v_i$. The local transformation that removes facet $\mathbf{a}v_i$ must be a type-N transformation since simplex $\mathbf{a}a_ka_{k+1}$ would be removed. Therefore, (D) is true for m. □

LEMMA 4. *If all interior facets of* $\mathcal{T}_i^{(r)}$ *not in* $\mathcal{G}_i^{(r)}$ *are locally optimal, then all interior facets in* $\mathcal{G}_i^{(r)}$ *are locally optimal.*

Proof. By Lemmas 1 and 3, all nonlocally optimal interior facets of $\mathcal{T}_i^{(r)}$ are in $\mathcal{F}_i^{(r)} \cup \mathcal{G}_i^{(r)}$. Suppose all interior facets in $\mathcal{F}_i^{(r)}$ are locally optimal. Suppose $\mathbf{a}v_i = a_1 \dots a_{k-1}v_i$ is a nonlocally optimal interior facet in $\mathcal{G}_i^{(r)}$. By Lemma 3, a type-N transformation is the only type of local transformation that can remove $\mathbf{a}v_i$. But a type-N transformation would also remove a nonlocally optimal interior facet from $\mathcal{F}_i^{(r)}$. This contradiction implies that $\mathbf{a}v_i$ is locally optimal. □

LEMMA 5. *Let* $\mathbf{a}a_k = a_1 \dots a_{k-1}a_k$ *be a nonlocally optimal and nontransformable interior facet in* $\mathcal{F}_i^{(r)}$ *incident on simplices* $\mathbf{a}a_ka_{k+1}$ *and* $\mathbf{a}a_kv_i$ *in* $\mathcal{T}_i^{(r)}$. *Suppose the nontransformability is due to* $(k-2)$*-face* $\mathbf{a}$, *i.e.,* $\mathbf{a}a_{k+1}v_i$ *is not a simplex of* $\mathcal{T}_i^{(r)}$, *but it is needed for the local transformation that removes* $\mathbf{a}a_k$ (*or* $\mathbf{a}$ *is in a configuration with* $\sigma_0 \geq 1$). *Then there exists another interior facet* $\mathbf{a}b$, $b \neq a_k$, *in* $\mathcal{F}_i^{(r)}$ *that is nonlocally optimal.*

Proof. Let the vertices in circular order about face $\mathbf{a}$ be $v_i, a_k, a_{k+1} = d_0, d_1, \dots, d_s = b, v_i$ where $s \geq 1$, i.e., $\mathbf{a}v_ia_k$, $\mathbf{a}a_kd_0$, $\mathbf{a}d_0d_1, \dots, \mathbf{a}d_{s-1}d_s$, $\mathbf{a}d_sv_i$ are simplices in $\mathcal{T}_i^{(r)}$. Since $\mathbf{a}d_0v_i$ is the missing simplex, $d_1, \dots, d_s$ must be on the same side of (the hyperplane containing) facet $\mathbf{a}d_0$ as v_i. By Lemmas 1 and 3, interior facets $\mathbf{a}d_0, \dots, \mathbf{a}d_{s-1}$ are locally optimal since they are not in $\mathcal{F}_i^{(r)} \cup \mathcal{G}_i^{(r)}$. The fact that $\mathbf{a}a_k$ is nonlocally optimal implies that $\bigcirc\mathbf{a}a_kd_0$ contains v_i. $\bigcirc\mathbf{a}d_0a_k$ does not contain d_1 so the part of $\bigcirc\mathbf{a}d_0d_1$ on the same side of $\mathbf{a}d_0$ as v_i contains the corresponding part of $\bigcirc\mathbf{a}d_0a_k$; in particular $\bigcirc\mathbf{a}d_0d_1$ contains v_i. This argument can be repeated to show that $\bigcirc\mathbf{a}d_1d_2, \ldots, \bigcirc\mathbf{a}d_{s-1}d_s$ all contain v_i. But the fact that $\bigcirc\mathbf{a}d_{s-1}d_s$ contains v_i implies that interior facet $\mathbf{a}d_s = \mathbf{a}b$ is nonlocally optimal, where $\mathbf{a}b$ is in $\mathcal{F}_i^{(r)}$. $\square$

To complete the proof that a Delaunay triangulation $\mathcal{T}_i^{(m)}$ is obtained for some m, we need to show that if $\mathcal{T}_i^{(r)}$ is not a Delaunay triangulation, then there exists a nonlocally optimal transformable interior facet in $\mathcal{F}_i^{(r)}$. This will be done in the next three lemmas, which use the following notations.

For a simplex $\mathbf{a}b = a_1 \ldots a_kb$, let $\mathbf{a}|b$ denote the half-space that consists of the hyperplane containing facet $\mathbf{a}$ as well as the points on the opposite side of this hyperplane from b. For two simplices $\mathbf{a}b$ and $\mathbf{a}c$ sharing common facet $\mathbf{a}$, $\mathbf{a}b$ is said to be *in front of* $\mathbf{a}c$ with respect to a viewpoint p if there exists a half-line L starting at p, such that L intersects the interior of both $\mathbf{a}b$ and $\mathbf{a}c$, and all intersections of L with the interior of $\mathbf{a}b$ are between p and any intersection of L with the interior of $\mathbf{a}c$. Note that if b and p are both in $\mathbf{a}|c$ and p does not lie on the hyperplane containing $\mathbf{a}$, then $\mathbf{a}b$ is in front of $\mathbf{a}c$ with respect to p. Also, for a point p and a simplex $a_1 \ldots a_{k+1}$, let $\phi_p(a_1 \ldots a_{k+1}) = \delta^2 - \rho^2$, where δ is the Euclidean distance between p and the centre of $\bigcirc a_1 \ldots a_{k+1}$, and ρ is the radius of $\bigcirc a_1 \ldots a_{k+1}$.

LEMMA 6. *Suppose $\mathcal{T}_i^{(r)}$ is not a Delaunay triangulation and all nonlocally optimal interior facets in $\mathcal{F}_i^{(r)}$ are nontransformable. Then there exists a cycle of simplices $a_1^0 \ldots a_{k+1}^0$, $a_1^1 \ldots a_{k+1}^1, \ldots, a_1^{t-1} \ldots a_{k+1}^{t-1}$, $a_1^t \ldots a_{k+1}^t = a_1^0 \ldots a_{k+1}^0$ in $\mathcal{T}_i^{(r)}$ such that v_i is not a vertex of any of these simplices, and for all j, $a_1^j \ldots a_{k+1}^j$ and $a_1^{j+1} \ldots a_{k+1}^{j+1}$ share common facet $a_1^j \ldots a_k^j$, $a_1^j \ldots a_k^j$ is locally optimal, and $v_i \in a_1^j \ldots a_k^j | a_{k+1}^j$.*

Proof. By Lemmas 1, 3, and 4 and Theorem 1, there exists a nonlocally optimal interior facet $\mathbf{a}a_k = a_1 \ldots a_{k-1}a_k$ in $\mathcal{F}_i^{(r)}$. By hypothesis, $\mathbf{a}a_k$ is nontransformable. Let $\mathbf{a}a_kv_i$ and $\mathbf{a}a_ka_{k+1}$ be the two simplices incident on $\mathbf{a}a_k$ in $\mathcal{T}_i^{(r)}$. Without loss of generality, suppose the nontransformability is due to $(k-2)$-face $\mathbf{a}$ as in the statement of Lemma 5. Then, by Lemma 5, $\mathbf{a}b_1 \in \mathcal{F}_i^{(r)}$ is also a nonlocally optimal interior facet where $b_1 \neq a_k$. By hypothesis, $\mathbf{a}b_1$ is also nontransformable. Suppose its nontransformability is due to $(k-2)$-face $\mathbf{b}$, where $\mathbf{b}$ consists of vertex b_1 and $k-2$ of the vertices $a_1, \ldots, a_{k-1}$ (since the angle at face $\mathbf{a}$ between facets $\mathbf{a}a_{k+1}$ and $\mathbf{a}v_i$ as viewed from b_1 is less than π, it is not possible that the nontransformability is due to face $\mathbf{a}$). Let $\mathbf{a}b_1 = \mathbf{b}b_k$, and $\mathbf{b}b_kv_i$ and $\mathbf{b}b_kb_{k+1}$ be the two simplices incident on $\mathbf{b}b_k$ in $\mathcal{T}_i^{(r)}$. There are two possible cases: either $a_{k+1} = b_{k+1}$ or $a_{k+1} \neq b_{k+1}$.

First suppose that $a_{k+1} = b_{k+1}$. Then $v_i \in \mathbf{a}a_{k+1}|a_k$ and by Lemmas 1 and 3, $\mathbf{a}a_{k+1}$ is locally optimal. Let $\mathbf{a}a_{k+1}a_k$, $\mathbf{a}a_{k+1}b_1$ be the start of a sequence of simplices. Now suppose $a_{k+1} \neq b_{k+1}$. Let the vertices in circular order about face $\mathbf{a}$ be v_i, a_k, $a_{k+1} = d_0, d_1, \ldots, d_s = b_{k+1}$, b_1, v_i where $s \geq 1$, i.e., $\mathbf{a}v_ia_k$, $\mathbf{a}a_kd_0$, $\mathbf{a}d_0d_1, \ldots, \mathbf{a}d_sb_1$, $\mathbf{a}b_1v_i$ are simplices in $\mathcal{T}_i^{(r)}$. By Lemmas 1 and 3, interior facets $\mathbf{a}d_0, \ldots, \mathbf{a}d_s$ are locally optimal. Since the nontransformability of $\mathbf{a}a_k$ is due to $\mathbf{a}$, v_i lies in the half-spaces $\mathbf{a}d_0|a_k$, $\mathbf{a}d_1|d_0, \ldots, \mathbf{a}d_s|d_{s-1}$. Let $\mathbf{a}d_0a_k$, $\mathbf{a}d_1d_0, \ldots, \mathbf{a}d_sd_{s-1}$, $\mathbf{a}d_sb_1$ be the start of a sequence of simplices.

The above argument can be repeated with $\mathbf{b}b_k$ and $\mathbf{b}$ taking on the role of $\mathbf{a}a_k$ and $\mathbf{a}$, and $\mathbf{b}c_1$ taking on the role of $\mathbf{a}b_1$, and so on, with more simplices added to the sequence of adjacent simplices. Since there are a finite number of facets in $\mathcal{F}_i^{(r)}$, there must be a cycle of adjacent nonlocally optimal nontransformable interior facets of $\mathcal{F}_i^{(r)}$. Without loss of generality, suppose this cycle of facets starts and ends at $\mathbf{a}a_k$. Then the proof of this lemma is completed by taking the sequence or cycle of adjacent simplices starting and ending at $\mathbf{a}a_{k+1}a_k$. □

To show that this cycle of simplices cannot occur, we need the following lemma from [6]. For completeness, we provide a simpler proof of this lemma here.

LEMMA 7. *Suppose* $\mathbf{a}b$ *and* $\mathbf{a}c$ *are simplices sharing common facet* $\mathbf{a} = a_1 \dots a_k$ *such that* $\mathbf{a}b$ *is in front of* $\mathbf{a}c$ *with respect to point* p*. If* c *is outside the circumhypersphere of* $\mathbf{a}b$*, then* $\phi_p(\mathbf{a}b) < \phi_p(\mathbf{a}c)$*.*

Proof. Suppose the coordinate system $(x_1, x_2, \dots, x_k)$ is first translated so that p is at the origin and then rotated so that the normal of facet $\mathbf{a}$ is $(0, \dots, 0, 1)$ and $\mathbf{a}$ lies on the hyperplane $x_k = \alpha_k$ with $\alpha_k > 0$. Note that this change to the coordinate system does not affect the value of ϕ_p. Let the radii of $\bigcirc\mathbf{a}b$ and $\bigcirc\mathbf{a}c$ be ρ_b and ρ_c, respectively, and let the centre of $\bigcirc\mathbf{a}b$ have coordinates $(\gamma_1, \dots, \gamma_{k-1}, \gamma_k)$ in the new coordinate system. Then the centre of $\bigcirc\mathbf{a}c$ has coordinates $(\gamma_1, \dots, \gamma_{k-1}, \gamma_k')$ with $\gamma_k' > \gamma_k$, since c is outside the circumhypersphere of $\mathbf{a}b$ and $\mathbf{a}b$ is in front of $\mathbf{a}c$ with respect to p. A formula for ρ_b^2 is

$$\rho_b^2 = (\alpha_1 - \gamma_1)^2 + (\alpha_2 - \gamma_2)^2 + \dots + (\alpha_k - \gamma_k)^2,$$

where $(\alpha_1, \dots, \alpha_{k-1}, \alpha_k)$ are the coordinates of a_1. A similar formula exists for ρ_c^2. By simple algebraic manipulation,

$$\phi_p(\mathbf{a}c) - \phi_p(\mathbf{a}b) = 2\alpha_k(\gamma_k' - \gamma_k) > 0. \qquad □$$

LEMMA 8. *If* $\mathcal{T}_i^{(r)}$ *is not a Delaunay triangulation, then there exists a nonlocally optimal transformable interior facet in* $\mathcal{F}_i^{(r)}$*.*

Proof. By Lemmas 1, 3, and 4 and Theorem 1, there exists a nonlocally optimal interior facet in $\mathcal{F}_i^{(r)}$. Suppose all nonlocally optimal interior facets in $\mathcal{F}_i^{(r)}$ are nontransformable. Then, by Lemma 6, there exists a cycle of simplices $a_1^0 \dots a_{k+1}^0$, $a_1^1 \dots a_{k+1}^1, \dots,$ $a_1^{t-1} \dots a_{k+1}^{t-1}$, $a_1^t \dots a_{k+1}^t = a_1^0 \dots a_{k+1}^0$ in $\mathcal{T}_i^{(r)}$ such that v_i is not a vertex of any of these simplices, and, for all j, $a_1^j \dots a_{k+1}^j$ and $a_1^{j+1} \dots a_{k+1}^{j+1}$ share common facet $a_1^j \dots a_k^j$, $a_1^j \dots a_k^j$ is locally optimal, and $v_i \in a_1^j \dots a_k^j | a_{k+1}^j$.

From the nondegeneracy assumptions made at the beginning of this section, $a_1^{j+1} \dots a_{k+1}^{j+1}$ is in front of $a_1^j \dots a_{k+1}^j$ with respect to $p = v_i$ for all j, and by Lemma 7,

$$\phi_p(a_1^0 \dots a_{k+1}^0) > \phi_p(a_1^1 \dots a_{k+1}^1) > \dots > \phi_p(a_1^t \dots a_{k+1}^t).$$

But the first and last values of this sequence of inequalities are identical. Therefore, this contradiction implies that the cycle does not exist, and there exists a nonlocally optimal transformable interior facet in $\mathcal{F}_i^{(r)}$. □

THEOREM 2. *If* $\mathcal{T}_{i-1}^D$ *is a Delaunay triangulation, then there exists a finite* $m \geq 0$ *such that* $\mathcal{T}_i^{(m)} = \mathcal{T}_i^D$ *is a Delaunay triangulation.*

Proof. By Lemma 3, the sequence of triangulations $\mathcal{T}_i^{(0)}, \mathcal{T}_i^{(1)}, \dots$ cannot cycle since each type-N transformation removes exactly one simplex that does not have v_i as a vertex and creates only simplices having v_i as a vertex. Hence there exists a finite m such that $\mathcal{T}_i^{(m)}$ is pseudolocally optimal. By Lemma 8, $\mathcal{T}_i^{(m)}$ is a Delaunay triangulation. □

5. Algorithms and data structure.

In this section, we describe three algorithms for constructing a k-D Delaunay triangulation based on the results of the previous section, as well as a data structure for representing the varying triangulations. These algorithms are k-D versions of the three-dimensional Delaunay triangulation algorithms in [13] and [15]. The first two algorithms differ in the order of adding the vertices to the triangulation. In the first, the n vertices are first sorted so that v_i is always outside the convex hull of $v_1, \ldots, v_{i-1}$. In the second, the vertices can be added in any order and a walk through adjacent simplices is used to determine the location of v_i in triangulation $\mathcal{T}_{i-1}^D$. The third algorithm does not explicitly perform the local transformations; they are made "implicit" so that nonlocally optimal nontransformable interior facets do not occur. Unlike the approaches in [29] and [28], a bounding simplex is not required in our algorithms.

The first and second algorithms are described by the following steps. By Lemmas 1, 3, and 4, only the facets of the $\mathcal{F}_i^{(r)}$ need to be tested for local optimality, so these facets are kept in a queue and the queue is updated as type-N transformations are performed.

ALGORITHM S

(1) *Initialization*: Sort $v_1, v_2, \ldots, v_n$ in lexicographic increasing order of their coordinates, and reorder the vertices slightly if necessary so that the first $k+1$ vertices are not in the same hyperplane. Form the first simplex $v_1 v_2 \ldots v_{k+1}$, and compute its centroid w. Initialize queue Q to be empty.

(2) For $i = k+2, \ldots, n$, add vertex v_i to $\mathcal{T}_{i-1}^D$, and apply local transformations to get $\mathcal{T}_i^D$ as follows:

   (a) Find a boundary facet $\mathbf{a}_0$ visible from v_i, i.e., v_i and w are on opposite sides of the hyperplane containing $\mathbf{a}_0$ (if v_{i-1} appears before v_i in lexicographic order, then there is a boundary facet containing v_{i-1} that is visible from v_i). Starting from $\mathbf{a}_0$, determine the adjacent boundary facets that are visible from v_i by checking neighbouring boundary facets of visible boundary facets. For each visible boundary facet $\mathbf{a}$, add simplex $\mathbf{a}v_i$ to the triangulation and insert $\mathbf{a}$ at end of queue Q.

   (b) While queue Q is not empty, remove the head facet $\mathbf{a}$ from Q and if $\mathbf{a}$ is still in the triangulation, do the following: Let $\mathbf{a}a_{k+1}$ and $\mathbf{a}a_{k+2}$, $a_{k+2} = v_i$, be the two simplices sharing interior facet $\mathbf{a}$. If $\bigcirc\mathbf{a}a_{k+1}$ contains v_i and $\mathbf{a}$ is transformable, then apply one or more local transformations (more than one may be needed if $\sigma_0 \geq 1$). For each local transformation, for each facet of $\mathcal{F}_B$ (as defined at the end of §2) that is an interior facet of the triangulation and does not have v_i as a vertex, add the facet to the end of Q.

ALGORITHM W

(1) *Initialization*: Reorder the vertices slightly if necessary so that the first $k+1$ vertices are not in the same hyperplane. Form the first simplex $v_1 v_2 \ldots v_{k+1}$, and compute its centroid w. Initialize queue Q to be empty.

(2) For $i = k+2, \ldots, n$, add vertex v_i to $\mathcal{T}_{i-1}^D$, and apply local transformations to get $\mathcal{T}_i^D$ as follows:

   (a) Starting from a simplex having v_{i-1} as a vertex, walk through neighbouring simplices of $\mathcal{T}_{i-1}^D$ to find a simplex or facet containing v_i (the location of v_i with respect to the facets of a simplex, as determined by barycentric coordinates, determines which adjacent simplex to visit next) or determine that v_i is outside the convex hull of $v_1, \ldots, v_{i-1}$ (the walk goes through a boundary facet $\mathbf{a}_0$). Lemma 7 guarantees that the walk does not cycle through simplices since each simplex is in front of the previous one with respect to v_i. If v_i is outside the convex hull, then create the new simplices, and add the visible boundary facets to queue Q as in Algorithm S, else delete and add the simplices described at the end of §3 and add the interior facets of $\mathcal{F}_i^{(0)}$ to Q.

   (b) This part is the same as in Algorithm S.

From §4, it can be seen that $\mathcal{T}_i^D = \mathcal{T}_i^{(m)}$ is obtained from $\mathcal{T}_{i-1}^D$ via type-N transformations that delete all simplices $a_1 \dots a_{k+1}$ in $\mathcal{T}_{i-1}^D$ whose circumhypersphere contains vertex v_i in its interior and add all simplices $a_1 \dots a_k v_i$, where $a_1 \dots a_k \in \mathcal{F}_i^{(m)} = \mathcal{F}_i^D$ (there are also some "temporary" simplices involving v_i that appear in the $\mathcal{T}_i^{(r)}$ but not in $\mathcal{T}_i^D$). Therefore, another approach is to not explicitly perform local transformations and record the temporary simplices of the form $a_1 \dots a_k v_i$, but to delete simplices $a_1 \dots a_{k+1}$ whose circumhypersphere contains v_i in its interior, record the facets that may form simplices with v_i, and create the simplices $a_1 \dots a_k v_i$ of $\mathcal{T}_i^D$ after the deletion is completed (as in Watson's algorithm [29]).

This "implicit" local transformation approach can be used in place of the explicit local transformations of Algorithm S or W. We provide the steps for Algorithm M below, where this algorithm is the "implicit" version of Algorithm W, since Algorithm W is faster than Algorithm S for sets of random points from the uniform distribution (see the next section). In Algorithm M, queue Q is used in a similar way to Algorithm W. In addition, stack S records the facets of $\mathcal{F}_i^D$. Algorithm M should be faster than Algorithm W, since local transformations are not explicitly performed and nonlocally optimal nontransformable interior facets do not occur. However, Algorithm W is more numerically robust than Algorithm M for the following reason. For sets of points on the same hypersphere, it is possible that round-off errors in determining whether a vertex is inside a hypersphere may cause an invalid triangulation with overlapping or missing simplices using the approach of Algorithm M in floating point arithmetic [29], [3], [13]. On the other hand, it is not possible for Algorithm W to produce invalid triangulations, since local transformations only change the simplices in a fixed local volume.

ALGORITHM M

(1) *Initialization*: Reorder the vertices slightly if necessary so that the first $k + 1$ vertices are not in the same hyperplane. Form the first simplex $v_1 v_2 \dots v_{k+1}$, and compute its centroid w. Initialize queue Q and stack S to be empty.

(2) For $i = k + 2, \dots, n$, add vertex v_i to $\mathcal{T}_{i-1}^D$ and delete and add simplices to get $\mathcal{T}_i^D$ as follows:

  (a) The walking phase of this part is the same as in Algorithm W. If v_i is outside the convex hull, then add the visible boundary facets to queue Q as in Algorithm S, else delete the simplices containing v_i from the triangulation and add the interior (boundary) facets of $\mathcal{F}_i^{(0)}$ to queue Q (stack S).

  (b) While queue Q is not empty, remove the head facet $\mathbf{a}$ from Q, and if $\mathbf{a}$ is still in the triangulation, do the following: Let $\mathbf{a}a_{k+1}$ be the remaining simplex incident on facet $\mathbf{a}$. If $\bigcirc\mathbf{a}a_{k+1}$ does not contain v_i, then push $\mathbf{a}$ onto S, else delete $\mathbf{a}a_{k+1}$ and $\mathbf{a}$ from the triangulation and for the other k facets of $\mathbf{a}a_{k+1}$, do the following: Let $\mathbf{b}$ be one of these other facets. If $\mathbf{b}$ is already in Q, then delete $\mathbf{b}$ from the triangulation; else if $\mathbf{b}$ is a boundary facet such that v_i lies on the hyperplane containing $\mathbf{b}$, then delete $\mathbf{b}$; else if $\mathbf{b}$ is a boundary facet, then push $\mathbf{b}$ onto S; else insert $\mathbf{b}$ at end of Q.

  (c) While stack S is not empty, pop the top facet $\mathbf{a}$ from Algorithm S, and add simplex $\mathbf{a}v_i$ to the triangulation.

As in the two-dimensional Delaunay triangulation algorithm in [28], the time in the walking phase of Algorithms W and M can be reduced by the following preprocessing bin sorting step (so that each vertex is likely to be closer to the previous one). Let $[x_{1,\min}, x_{1,\max}] \times \cdots \times [x_{k,\min}, x_{k,\max}]$ be the smallest hyperrectangle containing vertices $v_1, \dots, v_n$. This hyperrectangle is subdivided into $m^k \approx n^\alpha$ hyperrectangular bins of size $(x_{1,\max}-x_{1,\min})/m \times \cdots \times (x_{k,\max}-x_{k,\min})/m$, where α is chosen based on the distribution of points $v_1, \ldots, v_n$. (For example, for random points from the uniform distribution, $\alpha \approx 0.5$ and $\alpha \approx 0.4$ are optimal for two and three dimensions, respectively, [15].) The integers 1 to m^k are assigned to the bins such that consecutive integers correspond to adjacent bins, and the vertices are sorted based on their bin numbers, i.e., the vertices are inserted in increasing bin number order.

The data structure that we use to represent the varying triangulations in the three algorithms is the k-D version of the one for three-dimensional triangulations in [12], with the modified representation for boundary facets described in [13]. This data structure facilitates the walk through a triangulation, the determination of the neighbouring boundary facets of each boundary facet, the addition of new simplices involving v_i, the addition, deletion, and searching of facets, and the updates due to local transformations. The data structure consists of the following four arrays.

The vertex coordinates are stored in an array VC, where $VC[i].x_1$, $VC[i].x_2, \ldots$, $VC[i].x_k$ are the coordinates of the vertex v_i. The facets are stored in a hash table HT with direct chaining, where $HT[j]$ is the head pointer of the linked list of facets with hashing function value j. A new facet is added at the head of a linked list, since it is more likely to be referenced again. Let $i_1 < i_2 < \cdots < i_k$ be the k indices in VC of the k vertices of a facet. We use the hashing function $h(i_1, i_2, \ldots, i_k) = (i_1 n^{k-1} + i_2 n^{k-2} + \cdots + i_k) \bmod P$, where the hash table size P is a prime number that is approximately one-eighth of the number of facets in the final triangulation, since experimental results indicate that insertion, deletion, and searching operations take constant time, on average, with these choices.

We store the elements of the hash table linked lists in an array FC of facet records with origin index 1 (we use the array form of linked lists, which is more suitable for implementation in Fortran). The $k+4$ fields of $FC[j]$ are $i_1, i_2, \ldots, i_k, i_{k+1}, i_{k+2}|\, bfp$, $htlink$, $qlink$, where $0 < i_1 < \cdots < i_k$ are the k vertex indices of a facet; i_{k+1} and possibly i_{k+2} are the indices of the other vertex of the one or two simplices with (boundary or interior, respectively) facet $v_{i_1} \ldots v_{i_k}$; bfp is used for boundary facets only and is the negative of an index of the BF array described below (so the sign of $i_{k+2}|\, bfp$ indicates whether the facet is an interior or boundary one); $htlink$ is a pointer to the next element in the hash table linked list and is an index of FC or 0 (for end of list); $qlink$ is used by the algorithms to indicate whether or not the facet is in queue Q or some other list, and in the former case its value is a pointer to the next facet (i.e., an index of FC or 0), and in the latter case, its value is -1.

If a facet is in queue Q but is no longer in the triangulation, then the i_2 field is set to zero to indicate this and the facet record is deleted when it reaches the head of Q. We also use the i_1 field to maintain an available linked list of deleted facet records (the negative of the actual pointer value is stored to distinguish deleted facets from existing facets), so that a new facet record can be obtained from the available list if it is nonempty or the end of array otherwise.

The neighbouring boundary facets of each boundary facet are stored in an array BF of boundary records with origin index 1. The k fields of $BF[j]$ are $p_1, p_2, \ldots, p_k$, where each p_r is a pointer to a facet record (an index of FC) determined as follows. Let $i_1 i_2 \ldots i_k$ be a boundary facet stored in $FC[l]$ where the i_r are ordered vertex indices, and suppose the bfp field of $FC[l]$ is $-j$. Then the p_r field of $BF[j]$ is a pointer to the other boundary facet sharing the $(k-2)$-face $i_1 \ldots i_{r-1} i_{r+1} \ldots i_k$. As for FC, we also use the p_1 field of BF to maintain an available linked list of deleted boundary records. An example of the FC, HT, and BF arrays for a four-dimensional triangulation is given in Fig. 3.

FC	i_1	i_2	i_3	i_4	i_5	i_6| bfp	htlink	qlink
1	2	3	4	5	1	−1	0	−1
2	1	3	4	5	2	6	0	−1
3	1	2	4	5	3	6	0	−1
4	1	2	3	5	4	6	0	−1
5	1	2	3	4	5	−5	0	−1
6	2	3	5	6	1	−8	4	−1
7	1	3	5	6	2	4	5	9
8	1	2	5	6	3	4	1	−1
9	1	2	3	6	5	−7	3	14
10	2	4	5	6	1	−6	9	−1
11	1	4	5	6	2	3	6	−1
12	1	2	4	6	5	−2	2	−1
13	3	4	5	6	1	−3	12	−1
14	1	3	4	6	5	−4	8	0

HT	
0	7
1	11
2	10
3	13
4	14

BF	p_1	p_2	p_3	p_4
1	13	10	6	5
2	10	14	9	5
3	10	6	14	1
4	13	12	9	5
5	1	14	12	9
6	13	6	12	1
7	6	14	12	5
8	13	10	9	1

FIG. 3. *Illustration of FC, HT, and BF arrays for the four-dimensional triangulation containing the simplices* 12345, 12356, 12456, 13456 (*a simplex is represented by its vertex indices*). *There are three facets in queue with head* = 7 (*index of FC*).

In addition to the data structure operations, the main operations required are finding the hypersphere passing through the $k+1$ vertices of a simplex, finding the barycentric coordinates of a point with respect to the $k+1$ vertices of a simplex (this is used to determine the type of configuration and the direction of walking through adjacent simplices), and finding the hyperplane passing through the k vertices of a facet. The hypersphere and barycentric coordinates can each be found by solving a k-by-k system of linear equations, and the hyperplane can be found by solving a $k-1$-by-k system of homogeneous linear equations. We use Gaussian elimination with partial pivoting to solve these systems.

The three algorithms described above have been implemented in standard Fortran 77 (including all degenerate cases such as simultaneous degenerate local transformations), and this software has been added to GEOMPACK [15]. Relative tolerances are used throughout the implementation, e.g., for determining whether a point lies on a hypersphere or a hyperplane. The time complexities of the algorithms are discussed in the next section.

6. Time complexity.

6. Time complexity. We first derive the time complexity of Algorithm S, in the worst case and for sets of random points from the uniform distribution. Then we explain how these time complexities are different for Algorithms W and M. For simplicity, we assume that k is a constant in this analysis, although we will afterwards briefly discuss the factor of the time complexity that depends on k. We also present experimental results from our implementation of the three algorithms.

First we state some results from [5] and [27] on the worst-case size of convex hulls and Delaunay triangulations of n k-D vertices. The k-D convex hull has $O(n^{\lfloor k/2 \rfloor}/\lfloor k/2 \rfloor !)$ facets in the worst case. The k-D Delaunay triangulation contains $O(n^{\lceil k/2 \rceil}/\lceil k/2 \rceil !)$ simplices in the worst case. (Any k-D triangulation also contains $O(n^{\lceil k/2 \rceil})$ simplices in the worst case, since the simplices of a k-D triangulation become facets of a $(k+1)$-D convex hull if one extra $(k+1)$-D vertex is added, and $\lfloor (k+1)/2 \rfloor = \lceil k/2 \rceil$.) This means that the worst-case time complexity of any algorithm for constructing a k-D Delaunay triangulation is at least $O(n^{\lceil k/2 \rceil})$.

We now derive the worst-case time complexity of Algorithm S; this is a generalization of the three-dimensional case in [13]. $O(n \log n)$ time is required for the initial sorting of vertices in step 1. For step 2, we use the fact that using our hash table data structure with storage proportional to the maximum number of facets in the intermediate triangulations (which is usually close to the number of facets in the final Delaunay triangulation), searching, insertion, and deletion of a facet can each be performed in constant time, on average [16]. If $2k$ extra pointer fields are added to each facet record (one pointer to each other facet of each of the two simplices incident on the facet), then hashing and searching can be avoided and data structure operations can always be performed in constant time. The time complexity for step 2 can be obtained as follows.

For i between $k+2$ and n inclusive, let the following variables denote the quantities when vertex v_i is added to the triangulation:

BF_i = number of boundary facets in $\mathcal{T}_{i-1}^D$,

DF_i = number of facets in Q that are no longer in triangulation when facet is at head of Q,

LO_i = number of facets that result in locally optimal when tested,

NTF_i = number of facets that result in nontransformable when tested,

LT_i = number of local transformations applied,

DS_i = number of deleted simplices not involving vertex v_i,

AS_i = number of added simplices involving vertex v_i in $\mathcal{T}_i^D$.

Then the time complexity for step 2 is

$$O\left(\sum_{i=k+2}^{n}[BF_i + DF_i + LO_i + NTF_i + LT_i]\right). \tag{1}$$

The variables in (1) can be bounded as follows. $DF_i \le (k-1)LT_i$ since each type-N transformation removes at most $k-1$ facets of $\mathcal{F}_i^{(r)}$. Similarly, $NTF_i \le (k-1)LT_i$ since a nonlocally optimal nontransformable facet does not get put back in queue Q, but does get swapped out later by a type-N transformation or (simultaneous type-N transformations) involving another facet of the $\mathcal{F}_i^{(r)}$. $LO_i \le AS_i$ since every tested locally optimal facet forms a simplex with vertex v_i in $\mathcal{T}_i^D$. $LT_i = DS_i$ since each type-N transformation removes exactly one simplex not involving vertex v_i. Since each deleted simplex not involving vertex v_i must have been added at the insertion of a previous vertex, $\sum_{i=k+2}^{n} DS_i \le 1 + \sum_{i=k+2}^{n-1} AS_i$. With these bounds, (1) becomes

$$O\left(\sum_{i=k+2}^{n}[BF_i + AS_i]\right). \tag{2}$$

$BF_i = O(i^{\lfloor k/2 \rfloor})$ as mentioned above for the worst-case number of facets on a k-D convex hull. Also, $AS_i = O(i^{\lfloor k/2 \rfloor})$ for the following similar reason. Since all points on all facets of $\mathcal{F}_i^D$ are "visible" from v_i, the points of the facets can be linearly projected from v_i so that the projected facets are on the boundary of the convex hull of at most $i-1$ vertices (alternatively, project the facets onto a hyperplane and use the worst-case number of simplices in a $(k-1)$-D triangulation). Hence (2) and the worst-case time complexity of Algorithm S are $O(n^{\lfloor k/2 \rfloor + 1})$. Since $\lfloor k/2 \rfloor + 1 = \lceil k/2 \rceil$ for odd k, Algorithm S has a worst-case optimal time complexity for odd k.

For sets of i random k-D points from the uniform distribution (in a hypercube), the expected number of facets in the convex hull of the i points is $O((\log i)^{k-1})$ [4]. This size complexity for BF_i also occurs empirically, as seen in the following tables. Our measurements for AS_i show that it is also empirically $O((\log i)^{k-1})$. So from (2), the empirical time complexity of Algorithm S for sets of n random points from the uniform distribution is $O(n(\log n)^{k-1})$. The expected time complexity would also be $O(n(\log n)^{k-1})$ if it could be shown that the expected size of AS_i is the same as that for BF_i.

The derivation of the worst-case time complexity for Algorithms W and M is similar to that above, except there is one extra term WS_i for the number of simplices visited during the walk through $\mathcal{T}_{i-1}^{D}$. Since WS_i may be $O(i^{\lceil k/2 \rceil})$ in the worst case which is the same or worse than the $O(i^{\lfloor k/2 \rfloor})$ bounds for BF_i and AS_i in (2), the worst-case time complexity of Algorithms W and M may be $O(n^{\lceil k/2 \rceil + 1})$, which is worse than that for Algorithm S when k is odd (see the end of this section for further remarks on this). For sets of n random points from the uniform distribution, the time complexity of Algorithms W and M is harder to analyze (but it is the same for the two algorithms). Empirically, Algorithms W and M are faster than Algorithm S (see the tables below), because the bin sorting or some other preprocessing step can be used to reduce the time spent in the walking phase and, when v_i is added to the triangulation, there are about a constant number of local transformations applied when v_i is inside the convex hull of the previous vertices compared with about $O((\log i)^{k-1})$ local transformations applied when v_i is outside the convex hull.

As a function of k and n, the time complexities have an extra factor involving k that depends on implementation details. In our implementation, Gaussian elimination is used to solve k by k linear systems for hyperspheres, hyperplanes, and barycentric coordinates in $O(k^3)$ time per system. Also, each local transformation updates $O(k^2)$ facets in the data structure, and each insertion, deletion, or search operation in the hash table takes $O(k^2)$ time to order the k vertex indices of a facet (we assume k is small and use insertion sort for this); so it takes $O(k^4)$ time to update the data structure due to a local transformation. There is also an $O(k^4)$ factor from updating the BF array when v_i is added outside the convex hull of the previous vertices (since there are $O(k^2)$ $(k-2)$-faces in a simplex). Therefore, for the three algorithms, the extra factor in the time complexity is $O(k^4)$ (ignoring the constant factor for the convex hull size which is $1/\lfloor k/2 \rfloor!$ in the worst case and unknown for random points).

We now present experimental results from our Fortran 77 implementation of Algorithms S, W, and M, first for sets of random points from the uniform distribution and then for sets of points yielding a worst-case number of simplices in the Delaunay triangulation. The results are obtained from running our routines on a Sun 4/20 workstation using the *f*77 compiler with optimizing option. For each k, problems with four or five different values of n are used. For each n, five different problems are run and the measurements are averaged. Due to the increase in the CPU times and the number of facets as k increases, smaller sizes of n are used as k increases.

In Tables 1–4, for $k = 2$ to 5, are the CPU times in seconds for Algorithms S, W, and M (denoted by STIM, WTIM, and MTIM, respectively) and number of facets (NFAC) and boundary facets (NBF) in the Delaunay triangulation for sets of random points from the uniform distribution. For the average entries in the left half of the tables, at least three significant digits are given. The number of simplices is given by the formula (2 NFAC - NBF)$/(k+1)$. For $k = 2$ and 3, the bin-sorting preprocessing step was used with about $n^{0.5}$ and $n^{0.4}$ bins, respectively, for Algorithms W and M. For $k = 4$ and 5, the bin-sorting preprocessing step was not used since the n values are too small for it to be effective. These tables show that NFAC $= O(n)$, NBF $= O((\log n)^{k-1})$, and STIM $= O(n(\log n)^{k-1})$. Although Algorithm S has the better theoretical time complexity (because of no walking phase), Algorithms W and M are faster for the random points (with Algorithm M being the fastest because it does not apply local transformations explicity and has no nonlocally optimal nontransformable facets). Using larger n values and the bin-sorting preprocessing step for $k = 4$ and 5, we expect that the empirical time complexity of Algorithms W and M for $k = 4$ and 5 would be closer to that for $k = 2$ and 3, i.e., $O(n^{1.1})$.

TABLE 1
Average measurements for five random two-dimensional problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	$\frac{\text{NBF}}{\log_{10} n}$	$\frac{\text{STIM}}{n \log_{10} n}$	$\frac{\text{WTIM}}{n^{1.1}}$	$\frac{\text{MTIM}}{n^{1.1}}$
2000	5976	20.8	11.03	8.38	8.01	6.30	.00167	.00196	.00187
4000	11976	21.4	24.64	17.80	17.04	5.94	.00171	.00194	.00186
6000	17974	23.0	39.44	27.79	26.31	6.09	.00174	.00194	.00184
8000	23974	22.8	55.30	38.62	36.68	5.84	.00177	.00197	.00187
10000	29971	25.8	70.87	49.19	46.68	6.45	.00177	.00196	.00186

TABLE 2
Average measurements for five random three-dimensional problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	$\frac{\text{NBF}}{(\log_{10} n)^2}$	$\frac{\text{STIM}}{n(\log_{10} n)^2}$	$\frac{\text{WTIM}}{n^{1.1}}$	$\frac{\text{MTIM}}{n^{1.1}}$
1000	12762	143	44.06	29.08	23.20	15.9	.00490	.0146	.0116
2000	25978	174	105.21	62.99	50.17	15.9	.00483	.0147	.0117
3000	39237	193	172.33	98.67	77.52	16.0	.00475	.0148	.0116
4000	52635	210	241.60	134.51	106.33	16.2	.00466	.0147	.0116
5000	65980	204	315.90	171.31	135.41	14.9	.00462	.0146	.0116

TABLE 3
Average measurements for five random four-dimensional problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	$\frac{\text{NBF}}{(\log_{10} n)^3}$	$\frac{\text{STIM}}{n(\log_{10} n)^3}$	$\frac{\text{WTIM}}{n^{1.2}}$	$\frac{\text{MTIM}}{n^{1.2}}$
300	16682	530	68.20	52.35	35.50	34.9	.0150	.0558	.0378
600	37290	710	186.72	132.01	88.75	33.1	.0145	.0612	.0411
900	58287	874	330.56	218.45	146.15	33.9	.0142	.0623	.0417
1200	79841	1031	487.20	310.37	208.76	35.3	.0139	.0626	.0421
1500	101192	1186	672.41	400.06	270.33	37.0	.0140	.0618	.0417

TABLE 4
Average measurements for five random five-dimensional problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	$\frac{\text{NBF}}{(\log_{10} n)^4}$	$\frac{\text{STIM}}{n(\log_{10} n)^4}$	$\frac{\text{WTIM}}{n^{1.5}}$	$\frac{\text{MTIM}}{n^{1.5}}$
80	11879	942	41.80	40.61	24.39	71.8	.0398	.0568	.0341
160	35730	1597	169.40	148.77	86.96	67.7	.0449	.0735	.0430
240	61731	2258	343.25	291.90	168.13	70.3	.0446	.0785	.0452
320	90903	2869	558.16	448.61	260.34	72.8	.0443	.0784	.0455
400	119136	3565	800.99	630.37	363.65	77.8	.0437	.0788	.0455

Tables 1–4 illustrate how the CPU times increase for the k-D Delaunay triangulation algorithms as k increases. In [15], similar timing results are given for two- and three-dimensional versions of Algorithms S and W that work for one dimension only. Since the data structure and matrix operations in these versions do not have to work for arbitrary dimensions (for $k = 2$, a much simpler data structure can be used), these special versions are faster by the following factors. For the two-dimensional version, STIM is faster by

a factor of about 4.2 and WTIM is faster by a factor of about 5.0 over the k-D version run with $k = 2$. For the three-dimensional version, STIM and WTIM are both faster by a factor of about 1.6. We expect that special two- and three-dimensional versions of Algorithm M would have improvement factors similar to those of Algorithm W.

We use the following sets of points to obtain k-D Delaunay triangulations containing the worst-case number of simplices for $k \geq 3$ (all two-dimensional triangulations have $O(n)$ triangles). In [24], it is shown that if the n k-D points lie on the moment curve $(t, t^2, \ldots, t^k)$ with parameter t, then there exists a triangulation of the n points containing $O(n^{\lceil k/2 \rceil})$ simplices. Our routines show that the Delaunay triangulation of these points contains $O(n^{\lceil k/2 \rceil})$ simplices. In Tables 5–7, for $k = 3$ to 5, are the CPU times in seconds for Algorithms S, W, and M, and number of facets and boundary facets in the Delaunay triangulation for sets of random points on the moment curve, where the first coordinate t is a random number in the interval $[0, 1]$ from the uniform distribution. For Algorithms W and M, the n points are added to the triangulation in random order. For $k = 3$, NFAC $= (n-2)^2$ and NBF $= 2(n-2)$. For $k = 4$, NFAC $= (n-3)(3n-10)/2$ and NBF $= n(n-3)/2$. For $k = 5$, NFAC $= (n-3)(n-4)^2/2$ and NBF $= (n-3)(n-4)$.

TABLE 5
Average measurements for five three-dimensional moment curve problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	STIM/n^2	WTIM/n^2	MTIM/n^2
60	3364	116	1.33	2.61	2.51	.000369	.000724	.000696
120	13924	236	5.60	11.42	11.45	.000389	.000793	.000795
180	31684	356	12.81	26.35	26.77	.000395	.000813	.000826
240	56644	476	23.07	47.26	47.28	.000401	.000820	.000821
300	88804	596	37.37	75.08	75.98	.000415	.000834	.000844

TABLE 6
Average measurements for five four-dimensional moment curve problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	STIM/n^2	WTIM/n^2	MTIM/n^2
50	3290	1175	1.78	4.50	4.03	.000713	.00180	.00161
100	14065	4850	7.80	21.63	19.22	.000780	.00216	.00192
150	32340	11025	18.11	54.05	48.22	.000805	.00240	.00214
200	58115	19700	32.85	93.81	82.66	.000821	.00235	.00207
250	91390	30875	52.98	173.78	150.67	.000848	.00278	.00241

TABLE 7
Average measurements for five five-dimensional moment curve problems of each size.

n	NFAC	NBF	STIM	WTIM	MTIM	STIM/n^3	WTIM/n^3	MTIM/n^3
15	726	132	0.44	0.63	0.63	.000129	.000187	.000188
30	9126	702	5.68	9.95	9.09	.000210	.000369	.000337
45	35301	1722	22.17	40.37	36.42	.000243	.000443	.000400
60	89376	3192	57.77	104.40	98.15	.000267	.000483	.000454

Tables 5–7 show that the time complexity of Algorithms S, W, and M for the moment curve problems are each the same as the size complexity of the Delaunay triangulation, i.e., $O(n^{\lceil k/2 \rceil})$. Algorithm S is faster than Algorithms W and M for these problems, because no local transformations are performed by Algorithm S as a result of the lexicographic increasing order of the vertices while $O(n^{\lceil k/2 \rceil})$ local transformations are performed by Algorithm W.

TABLE 8
Average measurements for five worst-case time moment curve problems of each size.

$k=2$			$k=4$		
n	STIM	STIM/n^2	n	STIM	STIM/n^3
500	67.70	.000271	50	30.45	.000244
1000	277.68	.000278	100	274.32	.000274
1500	632.71	.000281	150	950.79	.000282
2000	1118.40	.000280	200	2274.26	.000284

For $k = 4$, Algorithm S does not achieve the worst-case time complexity of $O(n^3)$ for the moment curve problems in Table 6. This is due to the ordering of the vertices. If they are added in the reverse order, or equivalently, the first coordinate t is a random number in the interval $[-1, 0]$ from the uniform distribution, then the worst-case time complexity is achieved as shown in Table 8. This is because the addition of v_i causes $(i-4)(i-5)/2$ local transformations, $i = 6, \ldots, n$, for a total of $(n-3)(n-4)(n-5)/6$ local transformations. For $k = 2$, Algorithm S also achieves the worst-case time complexity of $O(n^2)$ for moment curve (or parabola) problems generated in the same way, as shown in Table 8. In this case, the addition of v_i causes $i-3$ local transformations, $i = 4, \ldots, n$, for a total of $(n-2)(n-3)/2$ local transformations. With the vertices ordered in the same way as Algorithm S, Algorithms W and M also achieve the worst-case time complexity for $k = 2$ and 4. For $k = 3$ and 5, the reverse ordering of vertices still results in no local transformations for Algorithm S.

For $k = 3$, Algorithms W and M could achieve a time complexity of $O(n^3)$ if the Delaunay triangulation of the first $i-1$ vertices contains $O(i^2)$ tetrahedra and the walk to determine the location of vertex v_i goes through $O(i^2)$ tetrahedra for all i. Edelsbrunner [7] has constructed the following example in which there is a line going through $O(n^2)$ tetrahedra in the Delaunay triangulation. For even n, the n vertices are $v_i = (0, -1+(i-1)h, 1)$, $i = 1, 2, \ldots, n/2$, and $v_i = (-1 + (i-1-n/2)h, (-1)^i\epsilon, 0)$, $i = n/2+1, \ldots, n$, where $h = 2/(n/2-1)$ and ϵ is a small positive number. The first $n/2$ vertices are on the line $x = 0$, $z = 1$. The last $n/2$ vertices are small perturbations of points on the line $y = 0$, $z = 0$. If ϵ is sufficiently small, then the Delaunay triangulation contains (at least) the $(n/2-1)^2$ tetrahedra $v_iv_{i+1}v_jv_{j+1}$ for $i = 1, \ldots, n/2-1$ and $j = n/2+1, \ldots, n-1$. The line $y = 0$, $z = \epsilon$ goes through the interior of each of these $(n/2-1)^2$ tetrahedra. However, a walk through the Delaunay triangulation to determine the location of a point p on this line does not have to go through all of the $(n/2-1)^2$ tetrahedra since the walk does not have to proceed in a straight line. From running our routines on this example, the walk actually goes through only $O(n)$ tetrahedra.

Therefore, this example cannot be used to obtain a time complexity of $O(n^3)$ for Algorithms W and M. So far, we have not been able to construct any examples for which the time complexity of Algorithms W and M is worse than $O(n^2)$ for $k = 3$. So it is an open question whether the worst-case time complexity of Algorithms W and M is actually worse than that of Algorithm S for odd k.

7. Concluding remarks. We have presented Algorithms S, W, and M for constructing Delaunay triangulations of sets of n k-D points. We have shown that the worst-case time complexity of Algorithm S is $O(n^{\lfloor k/2 \rfloor + 1})$, which is worst-case optimal for odd k. For sets of random points from the uniform distribution, the empirical time complexity of Algorithm S is $O(n(\log n)^{k-1})$.

Algorithms S and W both use local transformations, and only differ in the order of inserting the vertices into the incremental Delaunay triangulations. By allowing vertices to be inserted in the interior of the convex hull of the previous vertices, Algorithm W is faster than Algorithm S in practice, although Algorithm S has the better theoretical time complexity. Algorithm M differs from Algorithm W in that local transformations are not explicitly performed. So Algorithm M is faster than Algorithm W in practice, but is less numerically robust since invalid triangulations may be created by Algorithm M in floating point arithmetic when there are many points on (or nearly on) the same hypersphere. Using the criteria of efficiency and robustness, the best algorithm depends on the distribution and number of points to be triangulated.

In practice, we do not expect point distributions yielding a worst-case time complexity to occur. For example, in finite element mesh generation, the empirical time complexity will depend on how the points are generated, but should be close to that for the case of random points from the uniform distribution, e.g., see [14].

Shortly after submitting this paper, the author discovered similar work in [23] and [8]. In the former paper, it is shown that the incremental approach using local transformations can construct k-D Delaunay triangulations if the local transformations at each step are applied in a special order using a priority queue (instead of a more general topological order using a queue as in this paper). In the latter paper, which was written independently at about the same time as this one, a different approach is used to prove the results of §4 for regular triangulations (which include Delaunay triangulations as a special case).

Acknowledgment. The author would like to thank the referees for their helpful comments.

REFERENCES

[1] D. Avis and B. K. Bhattacharya, *Algorithms for computing d-dimensional Voronoi diagrams and their duals*, in Advances in Computing Research 1, F. P. Preparata, ed., JAI Press, Greenwich, CT, 1983, pp. 159–180.

[2] A. Bowyer, *Computing Dirichlet tessellations*, Comput. J., 24 (1981), pp. 162–166.

[3] J. C. Cavendish, D. A. Field, and W. H. Frey, *An approach to automatic three-dimensional finite element mesh generation*, Internat. J. Numer. Methods Engrg., 21 (1985), pp. 329–347.

[4] R. A. Dwyer, *On the convex hull of random points in a polytope*, J. Appl. Prob., 25 (1988), pp. 688–699.

[5] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[6] ———, *An acyclicity theorem for cell complexes in d dimensions*, in Proc. 5th ACM Symp. on Computational Geometry, ACM Press, New York, 1989, pp. 145–151.

[7] ———, *Private communication*, 1991.

[8] H. Edelsbrunner and N. R. Shah, *Incremental topological flipping works for regular triangulations*, in Proc. 8th ACM Symp. on Computational Geometry, ACM Press, New York, 1992, pp. 43–52.

[9] S. Fortune, *A sweepline algorithm for Voronoi diagrams*, Algorithmica, 2 (1987), pp. 153–174.

[10] A. Jameson and T. J. Baker, *Improvements to the aircraft Euler method*, in Proc. AIAA 25th Aerospace Sciences Meeting, Reno, NV, Paper 87-0452, 1987.

[11] B. Joe, *Delaunay triangular meshes in convex polygons*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 514–539.

[12] ———, *Three-dimensional triangulations from local transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 718–741.

[13] ———, *Construction of three-dimensional Delaunay triangulations using local transformations*, Comput. Aided Geom. Des., 8 (1991), pp. 123–142.

[14] ———, *Delaunay versus* max-min *solid angle triangulations for three-dimensional mesh generation*, Internat. J. Numer. Methods Engrg., 31 (1991), pp. 987–997.

[15] B. Joe, GEOMPACK – *a software package for the generation of meshes using geometric algorithms*, Adv. Engrg. Software, 13 (1991), pp. 325–331.

[16] D. E. Knuth, *The Art of Computer Programming*; *Vol.* 3 : *Searching and Sorting*, Addison-Wesley, Reading, MA, 1973.

[17] C. L. Lawson, *Software for C^1 surface interpolation*, in Mathematical Software III, J. R. Rice, ed., Academic Press, New York, 1977, pp. 161–194.

[18] ———, *Properties of n-dimensional triangulations*, Comput. Aided Geom. Des., 3 (1986), pp. 231–246.

[19] D. T. Lee and B. J. Schachter, *Two algorithms for constructing a Delaunay triangulation*, Internat. J. Comput. Inform. Sci., 9 (1980), pp. 219–242.

[20] C. S. Petersen, B. R. Piper, and A. J. Worsey, *Adaptive contouring of a trivariate interpolant*, in Geometric Modeling: Algorithms and New Trends, G. E. Farin, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 385–395.

[21] F. P. Preparata and S. J. Hong, *Convex hulls of finite sets of points in two and three dimensions*, Comm. ACM, 20 (1977), pp. 87–93.

[22] F. P. Preparata and M. I. Shamos, *Computational Geometry*, *An Introduction*, Springer-Verlag, New York, 1985.

[23] V. T. Rajan, *Optimality of the Delaunay triangulation in R^d*, in Proc. 7th ACM Symp. on Computational Geometry, 1991, ACM Press, New York, pp. 357–363.

[24] B. L. Rothschild and E. G. Straus, *On triangulations of the convex hull of n points*, Combinatorica, 5(1985), pp. 167–179.

[25] W. J. Schroeder and M. S. Shephard, *A combined octree/Delaunay method for fully automatic* 3-D *mesh generation*, Internat. J. Numer. Methods Engrg., 29 (1990), pp. 37–55.

[26] R. Seidel, *A convex hull algorithm optimal for point sets in even dimensions*, Tech. Rep. 81-14, Dept. of Computer Science, Univ. of British Columbia, 1981.

[27] ———, *On the number of faces in higher-dimensional Voronoi diagrams*, in Proc. 3rd ACM Symp. on Computational Geometry, ACM Press, New York, 1987, pp. 181–185.

[28] S. W. Sloan, *A fast algorithm for constructing Delaunay triangulations in the plane*, Adv. Engrg. Software, 9 (1987), pp. 34–55.

[29] D. F. Watson, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*, Comput. J., 24 (1981), pp. 167–172.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1437–1463, November 1993

A METHOD FOR DEVISING EFFICIENT MULTIGRID SMOOTHERS FOR COMPLICATED PDE SYSTEMS*

IRAD YAVNEH†

Abstract. A systematic approach is developed for gaining insight into complicated systems of partial differential equations (PDEs) in order to construct efficient smoothers for multigrid solvers. The method is derived from local mode (smoothing) analysis and employs algebraic graph theory, but it requires no knowledge of either in its implementation. It is applied to several problems, for which finding the best approach without such an analysis can be quite challenging.

Key words. multigrid, partial differential equations, systems

AMS subject classifications. 65M55, 65N55

1. Introduction. Multigrid methods are widely considered to be one of the most powerful and versatile approaches to the solution of problems with many variables, particularly discretized partial differential equations (PDEs) and systems. As more and more scientists and engineers become acquainted with these techniques, much of the mystery surrounding successful application of multigrid is lifted. And yet it is clear that the vast majority of multigrid solvers that are actually applied in scientific calculation are used for single equations or very simple systems. When multigrid methods are employed in truly complicated problems, they are usually applied only to small parts of the problem, e.g., some elliptic boundary value problem. Thus, the enormous potential of multigrid is rarely realized in practice. The main reason for this is that general tools that have been developed for dealing with complicated systems are frequently difficult to apply.

A method is introduced here for constructing the most important part of the multigrid solver, the smoother, for complicated systems. This method, which can be mastered readily by anyone capable of constructing a multigrid solver for a simple problem, frequently enables one to find a solution procedure that is as efficient as solving the equations of the system separately. Even when that is impossible, much is learned by applying this analysis, because it points out where the difficulties are.

Much of what is presented here has appeared in different form and from a different point of view in [1, §3]. When it exists, the equivalence between the two approaches will be noted.

A secondary purpose of this work is to suggest the use of graphs (via algebraic graph theory) in analyses. This may enable one, as here, to obtain tools that are far easier to utilize than the commonly used tools of linear algebra.

In §2 the analysis is presented along with examples. No mathematical justification is included in this section, but rather a heuristic motivation. In §3 the theoretical basis of the method is given, with more precise concepts. In §4 is a discussion of the main results, a comparison to present techniques and conclusions.

2. The coupling analysis. In constructing efficient relaxation schemes for multigrid solvers of discretized systems of PDEs, it is important to gain some understanding of the interrelationship between the equations. This is determined both by the differential

* Received by the editors May 18, 1992; accepted for publication (in revised form) December 14, 1992.This work was supported by the National Science Foundation through the National Center for Atmospheric Research.

† National Center for Atmospheric Research, Boulder, Colorado 80307-3000.

system and by the proposed relaxation scheme. The approach that motivates the analysis presented here is to consider the relaxation process as a feedback system. When an equation (or a block of equations) within a system is relaxed by some relaxation scheme, its residuals are decreased. But, due to the coupling between equations, the residuals of other equations in the system are perturbed. When these equations are relaxed in turn, such perturbations propagate and others are formed. Eventually, some of the perturbations caused by the relaxation of each equation may be fed back into the residuals of the equation, amplified by a certain factor that depends on the operators in the system (in particular, their orders and the size of their coefficients), on the discretization and relaxation scheme, and on the meshsize. If this factor is small enough for all the equations and all the possible routes such perturbations can follow, then the system behaves in relaxation almost as if it were decoupled. But if this is not the case, then the proposed relaxation process will generally fail.

As will be seen in §3, the same qualitative behavior is exhibited whether the equations are relaxed *simultaneously*, whereby the variables are updated only after all the equations have been relaxed, or *successively*, whereby the variables are updated as soon as each equation is relaxed.

The initial search for a useful relaxation scheme is greatly simplified by assuming a vanishingly small meshsize and finite coefficients of the operators. Then the feedback properties are determined by the orders of the operators in a straightforward manner, which is presented through the following example.

Suppose one wishes to construct an efficient, fully implicit, multigrid solver for the incompressible Euler equations in two dimensions, which can be written in terms of the vorticity $\zeta(x,y,t)$ and the streamfunction $\psi(x,y,t)$ as

$$\Delta\psi - \zeta = 0\,, \tag{1}$$

$$D_t\zeta = \zeta_t + J(\psi,\zeta) = 0\,, \tag{2}$$

where $J(\psi,\zeta) = \psi_x\zeta_y - \psi_y\zeta_x$, and Δ is the Laplace operator. Suppose that central differencing of some sort is used for the spatial discretizations, and Crank–Nicholson (or backward Euler) is used in time. Equation (1) is the Poisson equation for ψ, and it is well known (e.g., [1]) that point Gauss–Seidel (GS) relaxation, particularly in red-black (RB) ordering, provides an extremely efficient smoother. But GS relaxation for the advection operator D_t is useful only for small and moderate timesteps. When the timestep (more precisely, the Courant–Friedrichs–Lewy (CFL) number) becomes too large, point GS relaxation can no longer be used. It can be replaced by Kaczmarz relaxation (see [1, §1] and also §2.7 below), but this relaxation loses its efficiency very rapidly as the timestep is increased, and it is also more expensive. One might therefore attempt the following.

Consider the new variable $\Psi = \psi_t$. This leads to the following system, which is equivalent to (1)–(2).

$$\Delta\psi - \zeta = 0\,, \tag{3}$$

$$\zeta_t - \Delta\Psi = 0\,, \tag{4}$$

$$J(\psi,\zeta) - \Delta\Psi = 0\,. \tag{5}$$

Now, in the multigrid solution at each time-level, one can relax (3) by applying GS relaxation to ψ, relax (4) by applying GS relaxation to ζ, and (5) by similarly relaxing Ψ.

For each of these equations separately the relaxation is extremely efficient (indeed, GS relaxation applied to (4), considered as a function of ζ alone, is an exact solver). But, due to the coupling of the equations, relaxation of each equation affects the residuals of the others. Let us determine by analysis whether or not this effect of the coupling is prohibitive.

Given a system of k PDEs in k unknowns, the first step is to associate with each equation the variable for which the equation is relaxed. Thus, (3), (4), and (5) correspond to ψ, ζ, and Ψ, respectively. We shall later see what must be done if there is no such clear choice. The next step is to construct the k-by-k *order-array* Q: $q_{i,j}$, the element of Q in the ith row and jth column, is simply the order of the operator that operates on variable j in equation i. If variable j does not appear in equation i, then the null symbol N is placed in the (i,j)th position. For (3)–(5) the order-array is

$$Q = \begin{bmatrix} 2 & 0 & N \\ N & 1 & 2 \\ 1 & 1 & 2 \end{bmatrix}.$$

Note that the time-derivative is treated here similarly to the spatial derivatives. But there is an important distinction. We return to this in §2.3, §2.5, and §2.7.

From Q we next construct the k-by-k *weight-array* W: $w_{i,i}$, the diagonal elements of W, are the null symbol N; also, if $q_{i,j} = N$, then $w_{i,j} = N$; the remaining elements are given by

$$w_{i,j} = q_{j,j} - q_{i,j} ,$$

that is, the corresponding element in Q is subtracted from the diagonal Q element of the same column. For (3)–(5) W is given by

$$W = \begin{bmatrix} N & 1 & N \\ N & N & 0 \\ 1 & 0 & N \end{bmatrix}.$$

Now we plot the *coupling graph*. k nodes are plotted and numbered, each representing an equation. Then we go through the elements of W. If a null element is encountered it is skipped. But when $w_{i,j}$ is an integer, we draw a directed edge (an arrow) from node j to node i and label it with its weight $w_{i,j}$. The coupling graph for (3)–(5) is plotted in Fig. 1.

The final step in the analysis is to go over all possible simple loops (closed paths) in the coupling graph, moving only in the direction of the arrows, and sum up the weights along the loops. In Fig. 1 it is seen that only two such loops exist, (1,3,2,1) and (2,3,2), whose respective weight-sums are

$$s_{(1,3,2,1)} = w_{3,1} + w_{2,3} + w_{1,2} = 2$$

and

$$s_{(2,3,2)} = w_{3,2} + w_{2,3} = 0 .$$

DEFINITION 1. If the weight-sum of a loop is positive (nonpositive), then the equations whose nodes are included in the loop are said to be weakly (strongly) coupled.

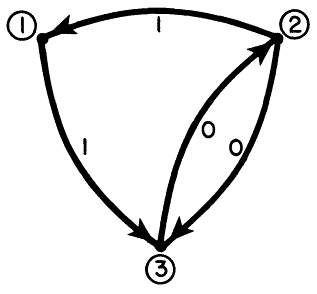


FIG. 1. *Coupling graph for* (3)–(5).

DEFINITION 2. If no loops exist, then the system is, in effect, decoupled. Otherwise, if the weight-sums of all the loops are positive, then the system is said to be weakly coupled.

Rule 1. If a system is weakly coupled, then relaxing its equations separately (successively) yields for the system a smoothing factor (i.e., the largest per-sweep amplification factor for high-frequency errors; cf. [1]) that tends to the largest (worst) of the smoothing factors of the operators on the diagonal of the system as the meshsize tends to zero.

This means that if a system is weakly coupled, then at least on fine enough grids it can be treated in relaxation as if decoupled, since the feedback is small for all the equations.

From the point of view of [1, §3], weak coupling implies that the principal part of the (linearized) matrix-operator is a diagonal matrix. Then, Brandt's principle that only the principal part of the operator needs to be relaxed (on fine enough grids) is automatically satisfied.

Rule 1 and the rules that follow are all derived from local mode (smoothing) analysis (see [1]) and hold under the same conditions as local mode analysis. But local mode analysis is well established as providing excellent prediction of smoothing performance, even when the assumptions on which it is based are not maintained exactly. In general, it can be said that the prediction holds so long as the coefficients of the (linearized) system are smooth on the scale of the mesh, i.e., their variation over a few meshsizes tends to zero with the meshsize. In practice, even if this condition is not satisfied over relatively small parts of the domain of solution, the smoothing efficiency is not affected significantly (see also [1].)

We find that (4) and (5) are strongly coupled. Therefore, these must generally be relaxed collectively, or else another formulation must be tried. Since the weight-sum of the corresponding loop is exactly zero, it is conceivable that relaxation may still be reasonably effective with this approach, depending on certain parameters. This is elaborated below. However, if the analysis had shown that the system were only weakly coupled, then the smoothing factor of the system would have been known to be the worst of the separate smoothing factors—that of the Laplace operator.

Nonlinear operators. As the first example indicates, the coupling analysis is not limited to linear systems. The order of a nonlinear operator is defined as follows.

Rule 2. The order of a nonlinear operator is determined by that of the linearized operator.

Thus, $J(\psi, \zeta)$ is a first-order operator for both ψ and ζ, and $(u_x)^3$ is a first-order operator for u, since the linearized operator is $3(u_x)^2\partial_x$. However, note that the order of a nonlinear operator is determined by the *discrete* operator, not the differential operator. Normally the orders are the same. But suppose, for example, that some nonlinear operator, say $J(\psi, \zeta)$, is discretized in conservation form,

$$J(\psi, \zeta) = \psi_x\zeta_y - \psi_y\zeta_x = (\psi_x\zeta)_y - (\psi_y\zeta)_x \, .$$

Although J is a first-order operator, this form includes second derivatives of ψ, which cancel each other in the differential operator. But they will only cancel each other in the discrete form if the discretizations of the two terms employ ψ-variables of the same meshpoints. If this is not satisfied by the discretization, then the operator is considered to be second-order in ψ for the purpose of the coupling analysis. Hence, in special cases such as will be examined later, one may choose to put off the decision of choice of discretization until after the analysis has been carried out.

2.1. Searching for weak-coupling configurations. Frequently, there is a natural and obvious correspondence between the equations in a system and the variables for which they should be relaxed. But, especially in complicated systems, there may be several plausible choices, and in other systems there may be no good choice at all. Treatment of the latter situation is examined in §2.5. Here we describe a general algorithm for finding a one-to-one correspondence between equations and variables that leads to a weakly coupled system, if it exists. Such a one-to-one correspondence is called a configuration, and a configuration that yields a weakly coupled system is called a weakly coupled configuration. Our convention is, as usual, that the the elements on the diagonal of the operator (and on the diagonal of the order-array Q) correspond to the unknowns for which the equations are relaxed.

The algorithm for searching for a weakly coupled configuration is as follows. Construct the order-array Q for some arbitrary configuration. Then find the permutation of the rows of Q that maximizes the trace (i.e., the sum of the diagonal elements), where the null symbol N is considered as $-\infty$. The following rule holds.

Rule 3. A configuration is weakly coupled if and only if the trace of its order-array Q is the unique maximum of the traces of all matrices obtainable by permutations of the rows of Q.

Hence, it suffices to find a permutation that maximizes the trace of Q, calculate the corresponding weight-array W, and construct the coupling graph. If the graph indicates that the system is weakly coupled, then it is the only weakly coupled configuration. But if the graph indicates strong coupling, then there exists no weakly coupled configuration for the formulation considered. An example of the application of this algorithm is given in §2.4.

The problem of maximizing the trace by row permutations is equivalent to the well-known *maximal* (*weighted*) *matching* problem, for which there are efficient algorithms (see, e.g., [5]). However, this problem is very easy by any means, unless the system is extremely large and complicated.

2.2. Coping with strong coupling. The following example illustrates what can be done when too strong a coupling is diagnosed. Suppose we wish to construct a smoother for the system

$$\Delta u - a^2 w = 0 \, , \tag{6}$$

$$\Delta v + \mathcal{M}(w) = 0 \, , \tag{7}$$

$$\Delta^2 u + J(u,v) + \Delta w = 0\,, \tag{8}$$

where $\mathcal{M}$ is some operator whose order q_M is yet to be determined, and a is a constant.

Let u, v, and w, respectively, correspond to (6), (7), and (8). The order-array is given by

$$Q = \begin{bmatrix} 2 & N & 0 \\ N & 2 & q_M \\ 4 & 1 & 2 \end{bmatrix},$$

and the weight-array by

$$W = \begin{bmatrix} N & N & 2 \\ N & N & 2 - q_M \\ -2 & 1 & N \end{bmatrix}.$$

The coupling graph is plotted in Fig. 2. The weight-sums of the loops are

$$s_{(1,3,1)} = w_{3,1} + w_{1,3} = 0$$

and

$$s_{(2,3,2)} = w_{3,2} + w_{2,3} = 3 - q_M\,.$$

The first and third equations are found to be strongly coupled. One way to solve this problem is to relax these equations collectively, but the resulting smoothing factor is uncertain and this form of relaxation is more expensive and complicated. It would be better if we could somehow sever the edge leading from the first equation to the third. Suppose we apply the Laplace operator to (6), subtract the resulting equation from (8) to obtain

$$J(u,v) + (1 + a^2)\Delta w = 0\,, \tag{9}$$

and use (9) instead of (8). The edge from the first equation to the third is not quite removed, but its weight increases from -2 to a very satisfactory 1, and the resulting system is weakly coupled provided that q_M is smaller than 3. A weakly coupled system has thus been obtained by *reducing the order of a crucial off-diagonal operator*, whose adverse effect was apparent in the coupling graph.

A more generally useful approach for coping with strong coupling is to transform the variables implicitly in such a way as to weaken the coupling. This is frequently the best approach, especially with staggered discretizations. It is a somewhat generalized form of Brandt's distributive relaxation (see, e.g., [3] or [1]). The generalization is that we do not necessarily require that the operator in the transformed variables form a triangular matrix. It suffices to obtain a weakly coupled system, which might ease the task of constructing the distributive scheme and lead to a less complicated smoother. This is especially important near boundaries, where ad hoc treatment of distributive relaxation is generally required (see, e.g., [4]). An example of this approach is given in §2.5.

Finally, if neither of these approaches is found to be useful, some or all of the equations might have to be relaxed collectively (once again relaxing the principal part of the operator, as suggested in [1, §3].) But even then the coupling analysis is useful in pointing out *which* equations are strongly coupled and therefore require such treatment.

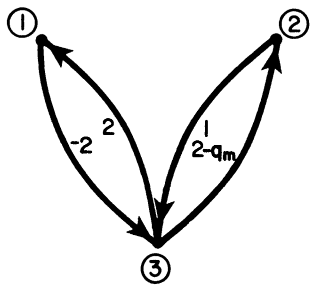


FIG. 2. *Coupling graph for* (6)–(8).

2.3. Finite meshsizes and the inclusion of coefficients. In applying the coupling analysis we have so far implicitly assumed that the coefficients in the equations were independent of the meshsize, and were therefore much larger (in absolute value) than the meshsize h and much smaller than $1/h$. This, however, need not be the case when finite meshsizes are considered, even in linear systems. A quantitative assessment of the strength of the coupling, which takes into account the meshsize and the coefficients, can be obtained just as easily. This includes, of course, the case of nonlinear systems, where the coefficients depend on the solution and its derivatives.

Rule 4. When the full coupling analysis is applied, the system must be written so that the coefficients of the operators on the diagonal (i.e., the operators of the variables that are relaxed) are $O(1)$.

If the coefficients of the operators on the diagonal are very large or very small, the equations must be divided through by these coefficients before the analysis is applied.

Consider the following linear elliptic system, whose solution is part of the explicit solution process of the shallow water balance equations [6], written in matrix form:

$$(10)\qquad \begin{pmatrix} \Delta + (R/H)\nabla\phi\cdot\nabla & 1/H & 0 \\ 0 & -\Delta & \Delta + 2R\mathcal{M} \\ R(A\Delta + \nabla\zeta\cdot\nabla) & 0 & \Delta \end{pmatrix} \begin{pmatrix} \chi \\ \Phi \\ \Psi \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix},$$

where ζ and ϕ are given functions, $\mathcal{M}$ is a second-order operator, R is a constant, which may be small or moderate, and H and A, which may vary considerably in size, are given functions of the spatial coordinates. Note that the system is written with $O(1)$ coefficients on the diagonal (indeed 1, but that is not essential).

The order-array and weight-array are, respectively,

$$Q = \begin{bmatrix} 2 & 0 & N \\ N & 2 & 2 \\ 2 & N & 2 \end{bmatrix}, \qquad W = \begin{bmatrix} N & 2 & N \\ N & N & 0 \\ 0 & N & N \end{bmatrix}.$$

The coupling graph is plotted in Fig. 3, and we find that there is a single loop, (1,3,2,1), whose weight-sum is 2. Hence, the system is weakly coupled.

In order to obtain a more quantitative assessment of the strength of the coupling, we now construct the *coefficient-array* C, whose element $c_{i,j}$ is some norm of the coefficients of the highest derivatives in the operator in equation i that operates on variable j. Here it is assumed that $c_{i,j}/h$ is not small compared to the norm of the coefficients of

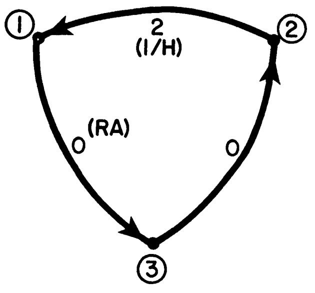


FIG. 3. *Coupling graph for* (10).

the second-highest derivatives in the corresponding operator, as may happen in singular perturbation problems. Otherwise, $c_{i,j}$, and also $q_{i,j}$, are defined as the coefficient-norm and order of the second-highest derivatives. Approximate values for $c_{i,j}$ are sufficient. For example, since R is known not to be large in (10), $c_{2,3} = 1$. The coefficient-array of (10) is

$$C = \begin{bmatrix} 1 & 1/H & N \\ N & 1 & 1 \\ RA & N & 1 \end{bmatrix} .$$

Now we return to the coupling graph and, for every element in C that is different from N (1's can also be disregarded), we write this element (in parentheses) on the edge leading from node j to node i. Finally, we go over all the loops and *multiply* the coefficients that are encountered. For (10) the coefficient-product of the only loop is

$$p_{(1,3,2,1)} = c_{1,2} \cdot c_{3,1} = RA/H .$$

In order to determine if the system is weakly coupled for a given meshsize h, we must add to the weight-sum of each loop the *log* of the loop's coefficient-product, divided by $\log h$, and still obtain a positive result. (This is due to the relative effects of coefficients and orders of the operators, and is clarified in §3). However, since now we are dealing with orders of magnitude, and with nonintegers, a more meaningful criterion is the following.

DEFINITION 3. Let s and p be the weight-sum and coefficient-product of a loop, respectively. Then the equations whose nodes are included in the loop are said to be h-weakly coupled if

$$ph^s \ll 1 .$$

DEFINITION 4. If all loops are h-weakly coupled, then the system is said to be h-weakly coupled.

Rule 5. If a system that is discretized on a grid with meshsize h is h-weakly coupled, then relaxing its equations successively yields for the system a smoothing factor

$$\bar{\mu} = \max_i \bar{\mu}_i + O(ch^\alpha) ,$$

where $\bar{\mu}_i$ are the smoothing factors of the operators on the diagonal of the system, and $c, \alpha > 0$, with c depending on the coefficients of the operators in the system.

Hence, if the meshsize is small, the equations can be treated in relaxation as if decoupled. Of course, the actual size of this positive power is of interest, and its calculation is explained in §2.6.

Sometimes (for example, in singular perturbation problems) the second-highest derivative terms may impose a greater restriction than the highest-derivative terms (because their coefficients are much larger). In such cases one can substitute some variables for the order and coefficient of the relevant operator when performing the coupling analysis and obtain conditions for weak coupling in terms of these variables.

Time derivatives. Consider once again the advection operator $D_t = \partial_t + u\partial_x + v\partial_y$. The coefficient of the spatial derivatives can be any rough estimate of the velocities u and v. In determining the coefficient of the time-derivative, however, we must multiply the actual coefficient (which is one in this case) by $h/\Delta t$, where Δt is the timestep. Thus, a good choice for the coefficient of D_t is, say, $h/\Delta t + |u| + |v|$. More generally, ∂_t^q is treated in the analysis as $(h/\Delta t)^q \partial_t^q$. Similar treatment must be used whenever spatial meshsizes differ significantly too. Returning to the first example, we find that (3)–(5) *is* h-weakly coupled only if $\Delta t(|\psi_x| + |\psi_y|)/h \ll 1$, i.e., at small CFL numbers, rather than at moderate and large ones for which the approach is required. This misfortune is not surprising, as at large CFL numbers the system loses its h-ellipticity (see [1]), and no point relaxation can be efficient. But the coupling analysis discovers this automatically, without requiring such insight.

Note the difference between time derivatives and spatial derivatives. In the solution process at each time level, the timestep Δt is fixed and only the spatial mesh is coarsened. This implies that the coupling of loops that include equations in which terms with time-derivatives are relaxed remains truly weak on coarse grids as well. The analysis predicts this phenomenon automatically. Such cases are considered in §2.5 and §2.7.

2.4. Coupling analysis of the shallow water balance equations. Let us see how the coupling analysis and the algorithm presented in §2.1 are employed to find an efficient fully implicit solver for a nonlinear, nonsymmetric coupled system—the shallow water balance equations (SWBE), which describe accurate, gravity-wave-free states on the so-called "slow manifold" of the shallow water equations. The full treatment of this problem is described in [6]. The nondimensionalized SWBE can be written for small or moderate Rossby number R and Burger number B as

$$-\nu\Delta\zeta + D_t\zeta + RA\Delta\chi = 0\,, \tag{11}$$

$$D_t\phi + H\Delta\chi = 0\,, \tag{12}$$

$$\zeta - \Delta\phi + 2RJ(\psi_x, \psi_y) = 0\,, \tag{13}$$

$$\zeta - \Delta\psi = 0\,, \tag{14}$$

where $D_t = \partial_t + J(\psi, \cdot) + R\nabla\chi \cdot \nabla$, $A = R^{-1} + \zeta$, and $H = B + R\phi$. Here, ν is a nondimensional viscosity coefficient, and in the following we consider for simplicity only the inviscid case of vanishing ν, which is the most difficult case with respect to smoothing. The system is discretized using a conservative central differencing in space and Crank–Nicholson in time.

Now we need to find a good smoother. Here it is not immediately obvious what, if any, effective correspondence between equations and variables exists. Following the algorithm of §2.1, we therefore construct the order-array for some configuration, and look for the permutation of rows that maximizes the trace. The maximal trace turns out to be seven, with (11), (12), (13), and (14) corresponding to ζ, χ, ϕ, and ψ, respectively. The order-array and weight-array are

$$(15)\qquad Q = \begin{bmatrix} 1 & 2 & N & 1 \\ N & 2 & 1 & 1 \\ 0 & N & 2 & q_M \\ 0 & N & N & 2 \end{bmatrix}, \qquad W = \begin{bmatrix} N & 0 & N & 1 \\ N & N & 1 & 1 \\ 1 & N & N & 2-q_M \\ 1 & N & N & N \end{bmatrix},$$

and the coefficient-array is

$$(16)\qquad C = \begin{bmatrix} 1 & RA\alpha & N & \alpha \\ N & 1 & 1/H\alpha & 1/H \\ 1 & N & 1 & R \\ 1 & N & N & 1 \end{bmatrix},$$

where q_M denotes the order of the discretized nonlinear operator in (13) and $\alpha = \Delta t/h$. Here, Rule 4 has been applied, and the coefficients in the advection operators were assumed to be $O(1)$. The coupling graph is plotted in Fig. 4. There are four loops, (1,3,2,1), (1,4,1), (1,4,2,1), and (1,4,3,2,1). The first three have weight-sums two, and the fourth $4 - q_M$. So the system is weakly coupled even if such a discretization is chosen that $q_M = 3$. The condition for weak h-coupling is that αh^2, $\alpha RA/Hh^2$, RAh^2/H, and $R^2 A h^{4-q_M}/H$ be much smaller than one.

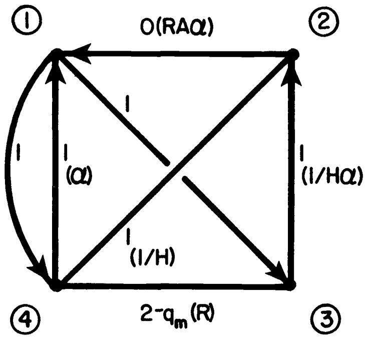


FIG. 4. *Coupling graph for* (11)–(14).

Thus it turns out that the SWBE can be smoothed efficiently simply by relaxing the equations separately.

2.5. Weakly coupled systems and distributive relaxation. As noted above, the coupling analysis requires that there be a *one-to-one* correspondence between each equation in the system and the variable for which it is relaxed. More generally, if some of the equations are relaxed collectively, then this block of equations is associated with the corresponding relaxed variables. But this is a condition that virtually *any* relaxation scheme must satisfy anyway. Sometimes there is no good choice. Consider, for example, the incompressible Euler equations in two dimensions in primitive variable form:

$$L_{IE}\begin{pmatrix} u \\ v \\ \phi \end{pmatrix} = \begin{pmatrix} D_t & 0 & \partial_x \\ 0 & D_t & \partial_y \\ \partial_x & \partial_y & 0 \end{pmatrix}\begin{pmatrix} u \\ v \\ \phi \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{17}$$

where $D_t = \partial_t + u\partial_x + v\partial_y$. The first and second equations can be relaxed for u and v, respectively, but the third does not even include ϕ. Brandt and Dinar (in [3]) and Brandt (in [1]) propose a highly efficient approach for dealing with this problem—distributive relaxation. This is a means of obtaining a weakly coupled system by an implicit transformation of variables, defined by a *distribution matrix* M. The transformed variables never truly appear, but are used to determine how changes must be made in the actual variables so that relaxation of one equation will have a small effect (at most) on the residuals of the others (see [1], [3], or [4] for details). For (17) the suggested distribution matrix M is given by

$$\begin{pmatrix} u \\ v \\ \phi \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\partial_x \\ 0 & 1 & -\partial_y \\ 0 & 0 & D_t \end{pmatrix}\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{\phi} \end{pmatrix}, \tag{18}$$

and the resulting system for the transformed variables is

$$L_{IE}M\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{\phi} \end{pmatrix} = \begin{pmatrix} D_t & 0 & 0 \\ 0 & D_t & 0 \\ \partial_x & \partial_y & -\Delta \end{pmatrix}\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{\phi} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{19}$$

This system is seen to be decoupled (as triangular operators yield no loops), with a straightforward correspondence between the transformed variables and the equations in which they are relaxed. Actually, the system is decoupled in a true sense only in the case of constant coefficients and periodic boundary conditions. Otherwise, as in the actual system here, the resulting system is only weakly coupled, since terms that cancel in the constant coefficient case do not cancel when the coefficients vary.

Triangularization of the transformed system is not the true object of the transformation, but rather, the transformed system needs to be weakly coupled (i.e., its principal part must be a diagonal matrix.) This is a more general goal. Suppose, for example, that we change the order in which the equations in (17) are written by exchanging the second and third rows of the matrix operator. Now, in order to obtain the same distributive relaxation, we must similarly permute the *columns* of the distribution matrix in (18), and the resulting transformed system, though weakly coupled, is no longer triangular.

There are no general rules for determining what distribution matrix is the simplest and most efficient. But the coupling graph provides a tool that frequently points to the best approach. Consider the nondimensional shallow water equations (SWE):

$$L_{SW}\begin{pmatrix} u \\ v \\ \phi \end{pmatrix} = \begin{pmatrix} D_t & -1/R & \partial_x \\ 1/R & D_t & \partial_y \\ \tilde{H}\partial_x & \tilde{H}\partial_y & D_t \end{pmatrix}\begin{pmatrix} u \\ v \\ \phi \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{20}$$

where $\tilde{H} = 1/F^2 + \phi$, F is the Froude number and R is the Rossby number. For this system,

$$Q = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \qquad W = \begin{bmatrix} N & 1 & 0 \\ 1 & N & 0 \\ 0 & 0 & N \end{bmatrix},$$

and the coupling graph (Fig. 5) shows that the first and third, as well as the second and third, equations are strongly coupled. The graph indicates that it suffices to raise the order of the diagonal operator of the third equation (without raising the order of the other operators). This can be done by applying the same distribution matrix M that is used for the incompressible Euler system. The resulting system for the transformed variables is

$$(21) \quad L_{SW}M \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{\phi} \end{pmatrix} = \begin{pmatrix} D_t & -1/R & \partial_y/R \\ 1/R & D_t & -\partial_x/R \\ \tilde{H}\partial_x & \tilde{H}\partial_y & (D_t)^2 - \tilde{H}\Delta \end{pmatrix} \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{\phi} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

The coefficient $\tilde{H}$ is known to be moderate or large (for physical reasons). Furthermore, assume that it is large compared to the square of the velocity, yielding an elliptic operator with a known good smoother for $\tilde{\phi}$ in the third equation (regardless of timestep size), this being the usual regime of solution. Hence, the coefficient of the operator on the diagonal of the third equation is approximately $\alpha^{-2} + \tilde{H}$, and we divide the third equation through by this coefficient. The *one-to-one* correspondence of variables and equations is now clear, and we can construct the order, weight, and coefficient arrays, obtaining

$$Q = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}, \quad W = \begin{bmatrix} N & 1 & 1 \\ 1 & N & 1 \\ 0 & 0 & N \end{bmatrix}, \quad C = \begin{bmatrix} 1 & \alpha/R & \alpha/R \\ \alpha/R & 1 & \alpha/R \\ 1 & 1 & 1 \end{bmatrix},$$

where $\alpha = \Delta t/h$, and the first two equations were multiplied by α in accordance to Rule 4. Here $c_{3,1}$ and $c_{3,2}$ are chosen to be one, rather than the more precise $\tilde{H}\alpha^2/(1+\tilde{H}\alpha^2)$, for simplicity. The latter term is bounded from above by one, and hence this choice represents the worst-case regime (since smaller off-diagonal coefficients weaken the coupling). But this is the typical case unless Δt is particularly small. The coupling graph of the transformed system is plotted in Fig. 6. The system can be seen to be h-weakly coupled under the condition

$$\Delta t/R \ll 1.$$

An efficient smoother is thus obtained that is virtually as simple as that of the incompressible Euler system.

Thus, while in (6)–(8) weak coupling was achieved by decreasing the order of an off-diagonal operator, the same end is reached here by *increasing*, in effect, the order of an operator *on* the diagonal. While here, as in (19), the principal part of the transformed system is triangular, this would not be the case if the equations were written in different order, and, as stated, it is not generally necessary.

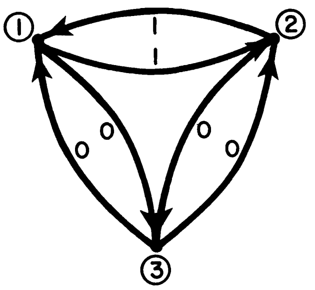


FIG. 5. *Coupling graph for* (26).

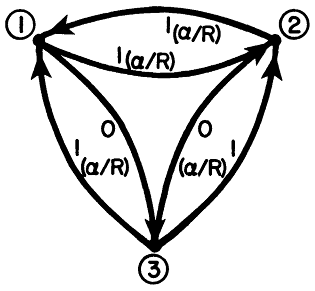


FIG. 6. *Coupling graph for* (27).

2.6. Estimating the weakness of coupling. An important notion, which is not fully addressed in Definition 4 and Rule 5, is the *degree* to which a given system is weakly coupled. This question requires a slightly more elaborate calculation.

Suppose that a given system of k partial differential equations in k unknowns is weakly coupled. Let $\bar{\mu}_i$ denote the smoothing factor corresponding to the ith equation. Then the following rule is used to calculate $\bar{\mu}$, the smoothing factor of the system.

Rule 6.

$$\bar{\mu} = \max_i \left[\bar{\mu}_i + O((ph^s)^{1/m})_i \right],$$

where

$$((ph^s)^{1/m})_i = \max_l \, (p_l h^{s_l})^{1/m_l(\bar{\mu}_i)}$$

with the maximum taken over all loops that include equations whose corresponding smoothing factor is approximately equal to $\bar{\mu}_i$ (i.e., tends to $\bar{\mu}_i$ as h tends to zero). Here, p_l is the coefficient-product of loop l, s_l is its weight-sum, and $m_l(\bar{\mu}_i)$ is the number of equations in loop l whose corresponding smoothing factors are approximately equal to $\bar{\mu}_i$.

We clarify this rule with examples. Consider the coupling graph in Fig. 3, which corresponds to system (10). There is only one loop, for which

$$ph^s = \frac{RA}{H} h^2 .$$

But in all three equations of the system the relaxed operator is the Laplacian, and all three are included in the loop. Therefore, by Rule 6, the smoothing factor of the system is

$$\bar{\mu} = \bar{\mu}_L + O\left(\left(\frac{RA}{H}h^2\right)^{1/3}\right),$$

where $\bar{\mu}_L$ is the smoothing factor for the Laplacian operator. So the smoothing factor of the system tends to that of the Laplacian only at the rate that $h^{2/3}$ tends to zero. This rate, however, can be improved (see §2.6.1 below).

Next, consider the coupling graph of (21), the shallow water system, which appears in Fig. 6. Now the relaxed operators of the first two equations are the same, but the third is different. There are three two-edge loops. Loop $(1,3,1)$ and loop $(2,3,2)$ have $ph^s = \Delta t/R$, but distinct $\bar{\mu}_i$'s. Loop $(1,2,1)$ does have two identical $\bar{\mu}_i$'s, but $ph^s = (\Delta t/R)^2$. Finally, there are two three-edge loops, $(1,2,3,1)$ and $(1,3,2,1)$, in which the advection operator again appears twice, but $ph^s = (\Delta t/R)^2$. Hence, the smoothing factor of the system differs from the worst of the smoothing factors of the individual equations ostensibly by only $O(\Delta t/R)$ on all the grids. (There is actually an $O(h)$ difference, due to the fact that the distribution matrix M yields exactly system (21) only if the coefficients are constant.)

2.6.1. Weakening weak coupling. Consider (10) once again. Suppose we multiply the first equation by RA, subtract the result from the third equation, and use this instead of the third equation (this manipulation is clearly "legal" for any finite values of R and A). For the new system

$$Q = \begin{bmatrix} 2 & 0 & N \\ N & 2 & 2 \\ 1 & 0 & 2 \end{bmatrix}, \qquad W = \begin{bmatrix} N & 2 & N \\ N & N & 0 \\ 1 & 2 & N \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 1/H & N \\ N & 1 & 1 \\ R(|\nabla\phi|RA/H + |\nabla\zeta|) & RA/H & 1 \end{bmatrix}.$$

The coupling graph is plotted in Fig. 7, ignoring coefficients for simplicity. It appears as if the strength of the coupling has not changed, since there is still a loop with weight-sum 2, but there is a subtle difference: this is a two-edge loop, and includes only two identical $\bar{\mu}_i$'s, whereas the loop of length three, which includes three identical $\bar{\mu}_i$'s, has a weight-sum of 3. So now the smoothing factor of the system tends to that of the Laplacian at the rate at which h (rather than $h^{2/3}$) tends to zero.

2.6.2. Weakening coupling by under-relaxation. When a system is h-strongly coupled, but only marginally so, a simple, yet frequently effective method for weakening the coupling is employing under-relaxation. In this well-known technique, the changes that are introduced in the relaxed variables are first multiplied by some factor $0 < \omega < 1$. This will frequently reduce the efficiency of relaxation for a single equation (though not always, e.g., Jacobi relaxation for the Poisson equation *requires* under-relaxation in order to be effective). But when the equation is part of a system, the perturbations its relaxation produces in other equations are reduced proportionally to the under-relaxation parameter.

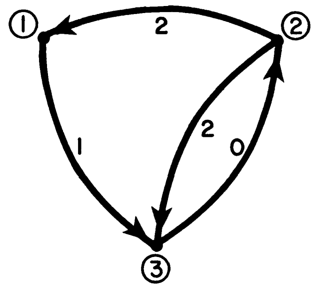


FIG. 7. *Coupling graph for modified* (10) *in* §2.6.1.

Suppose that, in a system of k equations, equation i, $i = 1, \dots, k$, is relaxed with an under-relaxation parameter ω_i. Then Rule 6 must be modified by multiplying each loop-product p by the product of the under-relaxation parameters ω_i corresponding to the equations participating in the loop. Also, $\bar{\mu}_i$ is now the smoothing factor of equation i with ω_i under-relaxation (see §3.3.1. for details.)

The under-relaxation parameters can be conveniently introduced in the coupling graphs by labeling the nodes. These results hold for over-relaxation ($\omega_i > 1$) as well, of course, but the coupling is then strengthened.

Employing under-relaxation may be useful especially on coarse grids, where h-weak coupling may be lost due to the relatively large meshsize (see [6].)

2.7. Numerical experiments. Numerical tests were performed to check various aspects of the coupling analysis. Far more extensive tests with the SWE and SWBE solvers will appear in separate reports. All experiments reported here were performed on a 2π-periodic square domain with uniform square mesh. The finest meshsize in all experiments was 128 by 128, and the coarsest 4 by 4, so that six levels were employed. The experiments were performed with the incompressible Euler equations in the formulations of (1)–(2) and (3)–(5), with system (10) (modified as suggested in §2.6.1), which appears in explicit solution of the SWBE system, with the fully implicit SWBE solver of (11)–(14), and with the SWE system (20). These are referred to as problems 1, 2, 3, and 4, respectively, below.

The multigrid cycle is defined by three parameters—the number of presweeps ν_1 (relaxation sweeps performed prior to each visit to the coarse grid), postsweeps (sweeps performed following coarse grid correction), and the cycle index, which is the number of times the next-coarser grid is visited before the correction of any intermediate grid is added to the next-finer grid solution [1]. We use the notation $\mathrm{V}(\nu_1, \nu_2)$ to denote V cycles (i.e., cycles with cycle index 1) with ν_1 presweeps and ν_2 postsweeps, and $\mathrm{W}(\nu_1, \nu_2)$ to similarly denote W cycles (whose cycle index is 2).

In all of the problems, initial conditions were prescribed on the finest grid, and a single step in time was performed, with multigrid cycles carried out until (fourteen-digit) roundoff errors were encountered. Here, problem 2 was tested within the framework of the SWBE explicit solver, but of course this has no effect on the results. The effective smoothing factor $\bar{\mu}$ was determined by calculating the asymptotic reduction factor per fine-grid relaxation sweep of the L_2 norms of the dynamic residuals averaged over all equations in the system.

In all problems, second-order Crank–Nicholson discretization was employed in time. In problems 1, 2, and 3, a conservation form was used for the spatial-derivative terms, with second-order central differencing on a nonstaggered grid (full details appear in [6]). In problem 4, nonconserving central differencing was used on a staggered grid, as in the steady-state incompressible Navier–Stokes solver described in [1], [3], and [4].

Where necessary, the average of the solution was subtracted from the solution before transferring residuals in order to maintain compatibility requirements necessary on the periodic domain (for example, existence of solution to (3) requires that the average ζ over the domain of solution be zero).

2.7.1. Incompressible Euler equations. Initial conditions for problem 1 were prescribed to be

$$\zeta(x, y, 0) = \sin(x)\ \sin(y) + rnd(-0.1, 0.1)\,,$$

where $rnd(a,b)$ denotes a uniformly distributed random number in the range (a,b). As noted in §2.3, the efficiency, indeed success or failure, of the proposed scheme (3)–(5) depends strongly on the CFL number, defined here by

$$\mathrm{CFL} = \max\ \Delta t(|\psi_x| + |\psi_y|)/h\,.$$

The numerical results confirm this. Employing V(1,1) cycles with GS relaxation in RB ordering, bicubic interpolation of corrections and full weighted residual transfers for all equations, the effective smoothing factor with CFL=1 was found to be 0.57, but already at CFL=2 there was no convergence at all. This of course is entirely unrelated to temporal stability considerations. The smoothing factors corresponding to formulation (1)–(2) with Kaczmarz relaxation in RB ordering for (2) were 0.25 in both cases since the smoothing factor of the Laplacian still dominates at CFL=2 (see problem 3 below).

2.7.2. System (10). System (10) was solved by V(1,1) cycles with GS relaxation in RB ordering and full weighted residual transfers. The effective smoothing factors were 0.25 with bicubic interpolation of errors and 0.30 with bilinear interpolation. The respective smoothing factors exhibited in the solution of the Poisson equation in seclusion were found to be 0.25 and 0.32. So the system performed as well as the Poisson equation, as predicted. (0.25 is of course the well-known smoothing factor predicted, e.g., in [1] for RB GS relaxation of the Poisson equation by precise smoothing analysis.) In these experiments the Rossby number R was 0.5, yielding RA/H in the range .37–2.2.

2.7.3. SWBE fully implicit solver. The fully implicit solver of the shallow water balance equations was tested under severe conditions by choosing the following form for the nonlinear term in (13):

$$J(\psi_x, \psi_y) = \frac{1}{2}\left[\nabla \cdot (\zeta \nabla \psi) - \frac{1}{2}\Delta(\nabla\psi \cdot \nabla\psi)\right]\,.$$

Hence q_M in (15) is three rather than two, and due to the introduction of first derivatives of ζ, $w_{3,1}$ in (15) is now zero rather than one. The main effect is that now the weight-sum of the loop (1,4,3,2,1) is one. Hence, by Rule 6, $\bar{\mu}$ of the system is given by

$$\bar{\mu} = \max\left[\bar{\mu}_L + O(h^{1/3}),\ \bar{\mu}_A + O(h)\right],$$

where $\bar{\mu}_L$ and $\bar{\mu}_A$ denote the smoothing factors of the Laplacian and the advection operator D_t, respectively.

The initial conditions for problem 3 were the same as for problem 1. A full account on how to obtain the other initial values is given in [6]. The tests were performed for the inviscid case, since the addition of viscosity leads to a smoothing factor that is at least as good. For all equations but the first, GS relaxation in RB ordering was employed. In the first equation, (11), Kaczmarz relaxation in RB ordering is used. In this relaxation the variables are scanned in the prescribed order, and the equation at each location is satisfied, in turn, by introducing to the variables changes that are proportional to the coefficients of these variables in the corresponding discretization stencil. This relaxation (unlike GS) remains stable at all CFL numbers, and is very efficient at moderate CFL numbers, provided that the viscosity coefficient is at most of the order of the meshsize. (Indeed, Kaczmarz relaxation for *steady state* (i.e., infinite CFL number) advection, discretized by first-order upstream discretization with no additional viscosity, is equivalent to GS relaxation of the Laplace operator when the streamlines are maximally nonaligned with the grid.) With greater viscosity, GS relaxation is efficient at all CFL numbers. In these equations the CFL number is defined by

$$\text{CFL} = \max\, \Delta t(|\psi_x + R\chi_y| + |\psi_y - R\chi_x|)/h\,.$$

Residuals were transferred by full-weighting, and both bilinear and bicubic interpolation of corrections were tested. R was again chosen to be 0.5, yielding a coefficient of one for the nonlinear term in (13), and B was prescribed to be one.

The results are listed in Table 1. At small CFL numbers, the 0.25 smoothing factor of the Laplacian dominates. At larger CFL numbers, relaxation of the advection operator becomes less efficient, and the corresponding performance of the advection equation in seclusion under similar conditions is included in Table 1 for V(1,1) cycles with bilinear interpolation for comparison.

TABLE 1
Effective smoothing factors exhibited by the implicit SWBE *solver, with some results of the advection-equation solver for comparison.*

CFL	Cycle	Interpolation	$\bar{\mu}_{\text{SWBE}}$	$\bar{\mu}_A$
1.0	V(1,1)	bicubic	0.26	
1.0	W(1,1)	bicubic	0.26	
1.0	V(1,1)	bilinear	0.36	0.21
1.0	W(1,1)	bilinear	0.29	
2.0	V(1,1)	bicubic	0.27	
2.0	W(1,1)	bicubic	0.25	
2.0	V(1,1)	bilinear	0.35	0.18
2.0	W(1,1)	bilinear	0.30	
3.0	V(1,1)	bicubic	0.30	
3.0	W(1,1)	bicubic	0.30	
3.0	V(1,1)	bilinear	0.33	0.31
3.0	W(1,1)	bilinear	0.30	
4.0	V(1,1)	bicubic	0.42	
4.0	W(1,1)	bicubic	0.42	
4.0	V(1,1)	bilinear	0.42	0.43
4.0	W(1,1)	bilinear	0.43	

The tests clearly indicate that the fully implicit solver of the SWBE performs as well as expected, yielding effective smoothing factors that are almost as good as that of the Laplacian at small CFL numbers and fully matching the performance of the advection-equation solver at larger CFL numbers. Indeed, given the severe conditions, the performance is somewhat better than the analysis might indicate. This is not unusual, as the analysis is based on a (quantitatively) slightly pessimistic view.

2.7.4. SWE solver. The SWE forms a system that is third-order in time, and therefore requires three initial conditions over the entire domain. In the tests these were given by

$$u(x,y,0) = \sin(x)\ \cos(y) + rnd(-0.1, 0.1)\,,$$

$$v(x,y,0) = -\cos(x)\ \sin(y) + rnd(-0.1, 0.1)\,,$$

$$\phi(x,y,0) = 0.25[\cos(2x) + \cos(2y)]\,.$$

The equations were solved with $F = 0.25$ and $R = 1$ and CFL numbers 1, 2, 3, and 4, defined here by

$$\mathrm{CFL} = \max\ \Delta t(|u| + |v|)/h\,.$$

V(1,1) cycles were employed with bilinear interpolation of corrections. The effective smoothing factors with CFL numbers 1–4 were 0.27, 0.26, 0.30, and 0.38, respectively. Comparison to Table 1 shows that the performance is as good as if the equations were decoupled.

2.7.5. Conclusions of numerical experiments. All the numerical experiments yielded results that conformed extremely well with the prediction of the coupling analysis. V cycles were found to suffice in order to obtain full efficiency. The reason for this is that, in equations with time derivatives, the CFL number is inversely proportional to the meshsize, so that the CFL number on coarse grids is much smaller than on the finest grid, leading to superior smoothing performance on the coarse grids. The steady-state type equations, on the other hand, are Poisson-like, and therefore V cycles easily suffice for these as well. Thus, in no case was any special treatment required on the coarse grids.

The smoothers for the SWE and SWBE are extremely efficient and straightforward to implement.

2.8. A systematic approach. We summarize the techniques introduced in this section by presenting a systematic approach to the development of an efficient relaxation scheme for a general system. This is done by taking the following steps.

1. Construct the order-array $\tilde{Q}$ for some arbitrary configuration.
2. Permute the rows of $\tilde{Q}$ so as to maximize the trace. Denote the resulting order-array by Q.
3. From Q construct the weight-array W and the coupling graph. Check for weak coupling.
4. If a weakly coupled system has been obtained, such that diagonal elements of Q correspond to operators that can be relaxed efficiently, then an efficient relaxation scheme for the system has been found. Check regime of weak coupling by including coefficients in the analysis (remember $O(1)$ coefficients for the diagonal operators).
5. If the resulting system is not weakly coupled, perform manipulations or implicit transformation of variables via a distributive relaxation scheme, so as to obtain a weakly coupled system. This is accomplished by raising the orders of key operators on the diagonal or lowering the orders of key off-diagonal operators. The key operators are determined from the coupling graph.
6. If a weakly coupled system cannot be obtained, relax equations that are strongly coupled collectively.

3. Theoretical development. The coupling analysis is derived from local mode analysis, which is the main tool for analyzing smoothing performance in particular, and multigrid processes in general. The full derivation is quite lengthy and will not be reported here. However, the main theorems are presented here with outlines of the proofs. Some acquaintance with smoothing analysis is presumed, but not with algebraic graph theory.

It should be noted that the full account of how to *implement* the coupling analysis is already described in §2.

3.1. Graphs and matrices. The basic concept upon which the coupling analysis is based is the following. Given a k-by-k matrix A, a directed graph G that describes A can be plotted. This is done by plotting k nodes, and for every nonvanishing element $a_{i,j}$ of A, connecting node j to node i with a directed edge (arrow), and labeling it with the "weight" $a_{i,j}$. (This convention is nonstandard with respect to the orientation, but it is convenient in the present application.) Here, j is referred to as the tail-node and i as the head-node. The diagonal terms of A correspond to edges that lead from the nodes to themselves and are treated similarly. As will be seen, certain matrix operations have useful meanings from the point of view of the corresponding graphs.

The following concepts will be used. A *path* is a concatenation of edges, such that the tail-node of each edge (except the first) coincides with the head-node of the preceding edge. A *simple loop* is a path whose every node is the tail-node of exactly one edge and the head-node of exactly one edge; this includes all single-edged loops unless noted otherwise. A *multiple loop* is a path that consists of two or more connected simple loops (i.e., simple loops with common nodes). The weight of a path is defined as the product of the weights of its edges.

LEMMA 1. *Let A be a k-by-k matrix and G its corresponding graph. Then each term in the determinant of A equals the product* (*or its negative*) *of the weight*(*s*) *of one or more disjoint simple loops in G, whose union includes every node in G exactly once. Conversely, every such product of loop-weights* (*or its negative*) *is a term in the determinant of A.*

Proof. Each term in the determinant of A is a product (or its negative) of k elements of A, no two of which share the same row or column. Hence, each node i is the tail-node of exactly one edge (corresponding to the element in the ith column) and the head-node of exactly one edge (corresponding to the element in the ith row). This proves the direct statement. Furthermore, every product of k elements from distinct rows and columns is a term (or its negative) in the determinant. This proves the converse.

3.2. Smoothing analysis. Consider a system of k linear, constant-coefficient PDEs in k unknowns,

$$Lu = \mathbf{f} \tag{22}$$

over $\mathcal{R}^n$, where $L = L(\partial_{\underline{x}})$ is a k by k matrix partial differential operator, $\mathbf{u}$ is the vector of unknowns, and $\mathbf{f}$ is the vector of forcing functions. Here, $\underline{x} = (x_1, \dots, x_n)$. All operators, variables, and functions are assumed to be in discretized form throughout the discussion.

Given some approximation $\tilde{\mathbf{u}}$ to $\mathbf{u}$, the error $\mathbf{v}$ is defined as the difference between the exact and approximate solutions,

$$\mathbf{v} = \mathbf{u} - \tilde{\mathbf{u}}\,. \tag{23}$$

Suppose that the system is discretized on a grid of meshsize h. Then $\mathbf{v}$ lends itself to the Fourier expansion

$$\mathbf{v}(\underline{x}) = \int_{-\pi}^{\pi} \hat{\mathbf{v}}(\underline{\theta}) e^{i\underline{\theta}\cdot\underline{x}/h} d\underline{\theta},$$

where $\underline{\theta} = (\theta_1, \ldots, \theta_n)$. $\mathbf{u}$ and $\tilde{\mathbf{u}}$ can be similarly expanded.

The purpose of relaxation in the multigrid process is to greatly reduce the amplitude of high-frequency components, defined as components satisfying

$$\pi/2 \leq |\underline{\theta}| \leq \pi,$$

where

$$|\underline{\theta}| = \max_{i=1}^{n} |\theta_i|.$$

If the relaxation of the equations is performed simultaneously, that is, each equation is relaxed using presweep values for all variables not corresponding to the relaxed equation, then

$$L_{new}\tilde{\mathbf{u}}_{new} + L_{old}\tilde{\mathbf{u}}_{old} = \mathbf{f}, \tag{24}$$

where

$$L_{new} + L_{old} = L.$$

Here, $\tilde{\mathbf{u}}_{new}$ and $\tilde{\mathbf{u}}_{old}$ are the approximate solutions after and before the sweep, respectively, and L_{new} and L_{old} constitute a partition of L that is determined by the particular relaxation that is applied. Equation (24) assumes that no over-relaxation or under-relaxation is used, but the generalization to such cases follows. Subtracting (24) from (22), and substituting (23), yields

$$L_{new}\mathbf{v}_{new} + L_{old}\mathbf{v}_{old} = 0,$$

and, hence,

$$\mathbf{v}_{new} = -L_{new}^{-1}L_{old}\mathbf{v}_{old}, \tag{25}$$

where $\mathbf{v}_{new}$ and $\mathbf{v}_{old}$ are the errors after and before the sweep, respectively.

In the following, it is convenient to consider the effect of relaxation in terms of reduction of the residual, $L\mathbf{v}$, rather than the error. Applying L to (25) produces

$$\begin{aligned} L\mathbf{v}_{new} &= -LL_{new}^{-1}L_{old}\mathbf{v}_{old} = -(L_{old} + L_{new})L_{new}^{-1}L_{old}\mathbf{v}_{old} \\ &= -(L_{old} + L_{old}L_{new}^{-1}L_{old})\mathbf{v}_{old} = -L_{old}L_{new}^{-1}(L_{new} + L_{old})\mathbf{v}_{old} \\ &= -L_{old}L_{new}^{-1}(L\mathbf{v}_{old}). \end{aligned} \tag{26}$$

The relaxation operator R is therefore given by

$$R = -L_{old}L_{new}^{-1}. \tag{27}$$

The smoothing analysis is mainly based on the assumption that the operators do not couple Fourier components, or couple only a small number of such components. Therefore, the components can be examined separately (or in small groups), with the difference-operators replaced by their *symbols*, defined for the operator L by

$$L(\partial_{\underline{x}})e^{i\underline{x}\cdot\underline{\theta}/h} = \hat{L}(i\underline{\theta}) \cdot e^{i\underline{x}\cdot\underline{\theta}/h}. \tag{28}$$

Here, the matrix $\hat{L}$, whose elements are generally polynomials in exponentials of $i\theta_i$, is the symbol of L. The symbol of the relaxation operator R is denoted by $\hat{R}$. If relaxation is such that there is no coupling between components, as in Jacobi relaxation, then $\hat{R}$ is a k-by-k matrix, but if several components are coupled, as in GS relaxation in RB ordering, then the dimension of R is multiplied by the number of coupled components. Such cases are not considered here.

The *smoothing factor* $\bar{\mu}$ corresponding to $\hat{R}$ is its maximal spectral radius ρ over all the high frequencies. Here, $\rho = \rho(\underline{\theta})$ is the largest of the absolute values of the eigenvalues. A successful relaxation scheme must obviously have a smoothing factor that is significantly smaller than one. Also, the spectral radii for *all* components can be assumed to be at most one (this condition can be weakened slightly, but the distinction is unimportant here).

Note that the smoothing properties in terms of the relaxation operator defined by the residual-reduction are the same as those defined by the error-reduction, since their eigenvalues are the same. (For any square matrices A and B, if $AB \cdot v = \lambda v$, multiplying by B yields $BA \cdot Bv = \lambda Bv$, proving that AB and BA have the same nonvanishing eigenvalues. For vanishing eigenvalues the result is trivial.)

3.3. Weakly coupled systems. Suppose that L is such that to each equation in system (22) there corresponds a variable for which it is relaxed, having a satisfactory smoothing factor for that equation in seclusion. The equations and their corresponding variables (and smoothing factors) will be denoted by the same subscripts. Also, let the equations be written so that the coefficients of the operators on the diagonal are all $O(1)$. There are generally two ways of relaxing the equations: simultaneously, whereby all the equations are relaxed, and only then the approximate solution is updated; and successively, whereby the equations are relaxed in some prescribed order, and the approximate solution is updated as soon as each equation is relaxed. The former is simpler to analyze and shall be considered first. Also, the discussion is simplified by only considering relaxation that does not couple components. The generalization to schemes that do couple a few components is straightforward (and all the numerical results in §2.7 were obtained with such schemes). Similarly, we do not consider here the case where some of the equations are relaxed collectively. Therefore, all the elements of $\hat{L}_{new}$ and $\hat{L}_{old}$, except elements on the respective diagonals, are either the elements of $\hat{L}$, which shall be denoted by $l_{i,j}$, or zero. The diagonal elements of $\hat{L}_{new}$ are denoted by n_i, and those of $\hat{L}_{old}$ by o_i. The elements of $\hat{R}$ are denoted by $r_{i,j}$. $q_{i,j}$ denotes the order of the operator in L in the (i, j)th position, and $c_{i,j}$ denotes some norm of the coefficients of the highest derivatives in this operator. Here it is assumed that $c_{i,j}/h$ is not small compared to the norm of the coefficients of the second-highest derivatives in the corresponding operator. Otherwise, $c_{i,j}$ and $q_{i,j}$ are defined as the coefficient-norm and order of the second-highest derivatives.

3.3.1. Simultaneous relaxation. When the relaxation is performed simultaneously, $\hat{L}_{new}$ is a diagonal matrix, since only old values are used for all the unknowns except those relaxed. Hence, by (27), $\hat{R}$ is given by

$$r_{i,j} = \begin{cases} o_j/n_j \equiv \mu_i & \text{if } i = j, \\ l_{i,j}/n_j & \text{otherwise,} \end{cases} \tag{29}$$

where $\mu_i(\underline{\theta})$ is by definition the reduction factor for an error Fourier component of frequency $\underline{\theta}$ corresponding to the ith equation in seclusion. (The smoothing factor, $\bar{\mu}_i$, is the maximal of the absolute values of μ_i over all the high frequencies.)

Now, a consistent discretization and the condition that $\bar{\mu}$ (and similarly $\bar{\mu}_i$) be smaller than one imply

$$n_i = O(h^{-q_{i,i}}). \tag{30}$$

For the off-diagonal terms in $\hat{R}$, this yields

$$r_{i,j} = O(c_{i,j}h^{q_{j,j}-q_{i,j}}), \qquad i \neq j, \tag{31}$$

since $l_{i,j}$ is at most $O(c_{i,j}h^{-q_{i,j}})$. (Indeed, it is only as large as this for high-frequency components and is only $O(c_{i,j})$ for smooth components.)

The eigenvalues of $\hat{R}$ are the roots λ_i, $i = 1, \ldots, q$, of the characteristic equation

$$|\hat{R} - \lambda I| = 0. \tag{32}$$

Rewrite (32) in the form

$$\prod_{j=1}^{k} (\mu_j - \lambda) = rhs, \tag{33}$$

where all the terms but the product of the diagonal terms of $\hat{R} - \lambda I$ have been transferred to the right-hand side. Equation (33) implies the following.

LEMMA 2. *If $rhs = O(ch^\alpha)$ for some positive c and α, then the eigenvalues of $\hat{R}$ satisfy*

$$\lambda_i = \mu_i + O((ch^\alpha)^{1/m_i}), \tag{34}$$

where m_i is the multiplicity of μ_i, defined as the number of diagonal elements of $\hat{R}$ that tend to μ_i as $h \to 0$.

Proof. By (33),

$$(\mu_i - \lambda_i)^{m_i} \leq \tilde{c}ch^\alpha$$

for some constant $\tilde{c}$, from which the proof follows.

The effect of the multiplicity is depicted in the following example. Let

$$\hat{R} = \begin{pmatrix} \mu_1 & r_{1,2} \\ r_{2,1} & \mu_2 \end{pmatrix}.$$

The characteristic equation is

$$(\mu_1 - \lambda)(\mu_2 - \lambda) - r_{1,2}r_{2,1} = 0,$$

and its roots are

$$\lambda_{1,2} = \frac{1}{2}\{\,(\mu_1+\mu_2) \pm [\,(\mu_1-\mu_2)^2 + 4r_{1,2}r_{2,1}\,]^{1/2}\,\}\,.$$

Hence, $\lambda_j = \mu_j + O(r_{1,2}r_{2,1})$ if $\mu_1 - \mu_2 = O(1)$, but $\lambda_j = \mu_j + O(\sqrt{r_{1,2}r_{2,1}})$ if $\mu_1 - \mu_2 \approx 0$.

Consider now the graph G corresponding to $\hat{R} - \lambda I$ or, equivalently, to $\hat{R}$.

LEMMA 3. *If the maximal loop-weight over all the multi-edged simple loops of G is $O(ch^\alpha)$ for some positive c and α, then the eigenvalues of $\hat{R}$ satisfy*

$$\lambda_i = \mu_i + O((ch^\alpha)^{1/m_i})\,.$$

The proof follows from Lemma 1 and Lemma 2. This is of course the h-weak coupling of the system defined in Definition 4, and Rule 5 follows. Rule 1 also follows by fixing the coefficients and sending h to zero. Rule 6, however, and indeed the concept of the strength of coupling within *subsets* of equations, requires a closer look at rhs.

THEOREM 4. *Let w_l denote the weight of simple loop l, in a system for which all loops satisfy $1 > w_l = O(ch^\alpha)$ for some positive c and α. Then the eigenvalues of $\hat{R}$ satisfy*

$$\lambda_i = \mu_i + O\left(\max_l\,(w_l)^{1/m_i^{(l)}}\right), \tag{35}$$

where the maximum is taken over all multi-edged simple loops, and $m_i^{(l)}$ is the multiplicity of μ_i in loop l, defined as the number of equations in loop l whose corresponding diagonal element in $\hat{R}$ tends to μ_i as $h \to 0$.

Sketch of Proof. Representing each group of μ_j's that tend to the same value for vanishing meshsize by $\tilde{\mu}_j$, we have from (33)

$$\prod_{\tilde{\mu}_j}(\tilde{\mu}_j - \lambda)^{m_j} = c_1\sum_j rhs_j = c_2 rhs_k\,,$$

where rhs_j are the terms of rhs, and $rhs_k = \max_j rhs_j$. Here and below c_i denote constants. Now, fix $\lambda = \lambda_i$, where λ_i is the eigenvalue of $\hat{R}$ that tends to μ_i as the meshsize tends to zero (following Lemma 3). Assume $\lambda_i = \mu_i + O(\epsilon)$. Then

$$\epsilon^{m_i} = c_3 rhs_k\,.$$

Let $\mathcal{L}_k$ denote the set of all multi-edged loops in rhs_k, and let $s_{i,k}$ denote the number of single-edged loops in rhs_k corresponding to $\tilde{\mu}_i$. Then, by Lemma 1, $rhs_k = c_4\epsilon^{s_{i,k}} \prod_{l\in\mathcal{L}_k} w_l$. Hence,

$$\epsilon^{m_i - s_{i,k}} = c_5 \prod_{l\in\mathcal{L}_k} w_l\,.$$

But $(m_i - s_{i,k}) = \sum_{l\in\mathcal{L}_k} m_j^{(l)}$ so that

$$\epsilon = c_6\left(\prod_{l\in\mathcal{L}_k} w_l\right)^{1/(m_i - s_{i,k})} \le c_6 \max_{l\in\mathcal{L}_k}(w_l)^{1/m_j^{(l)}} \le c_6 \max_l (w_l)^{1/m_j^{(l)}}\,.$$

The simple proof of the inequality used here is omitted for brevity.

Note that since, by Lemma 1, *all* products of disjoint simple loops that include all the nodes appear in rhs, one of the terms must have the loop with the maximal $w_l^{1/m_i^{(l)}}$ as its *only* multi-edged loop-weight factor; and, therefore the estimate of Theorem 4 is the best general estimate possible. Rule 6 follows directly from Theorem 4.

Under-relaxation. Let Ω be a diagonal matrix whose elements $\Omega_{i,i}$ are the respective under-relaxation (or, equivalently, over-relaxation) parameters employed. $\Omega = I$ (the identity matrix) implies that no under-relaxation is applied. When under-relaxation is employed, (27) is replaced by

$$R_\Omega = I + \Omega(R - I)\,, \tag{36}$$

where $R = R_I$ is as defined in (27). Following the proof of Theorem 4, it is now straightforward to generalize this theorem to the case where under-relaxation is employed, obtaining

$$\lambda_i = 1 - \Omega_{i,i} + \mu_i\Omega_{i,i} + O\left(\max_l \left(w_l \prod_l \Omega_{j,j}\right)^{1/m_i^{(l)}}\right), \tag{37}$$

where the product is taken over the under-relaxation parameters corresponding to the equations included in loop l.

3.3.2. Derivation of Rule 3. To outline the proof of Rule 3, let $\tilde{L}$ be a matrix whose elements are

$$\tilde{l}_{i,j} = l_{i,j}/l_{j,j}\,. \tag{38}$$

Then, by (29)–(33), the weak-coupling condition of Lemma 2, viz., $rhs = O(ch^\alpha)$, is satisfied if and only if

$$|\tilde{L}| = 1 + O(ch^\alpha)\,, \tag{39}$$

barring freak cancellations that are neglected throughout this work and are discussed in §4. But, by (38),

$$|\tilde{L}| = |\hat{L}| / \prod l_{j,j}\,, \tag{40}$$

where the product is taken over all the diagonal elements of $\hat{L}$. In orders of magnitude of h, this yields

$$|\tilde{L}| = |\hat{L}| \cdot O\left(h^{\sum_j q_{j,j}}\right). \tag{41}$$

Now $|\hat{L}|$ is invariant under row permutation, so $|\tilde{L}|$ is minimal when the sum of the orders $q_{j,j}$ of the diagonal operators of L is maximal. Furthermore, $\lim_{h\to 0} \min |\tilde{L}| \geq 1$, where the minimum is taken overall row permutations of $|\hat{L}|$, since the diagonal elements of $\tilde{L}$ are all ones. Hence, if the sum of the orders of the diagonal elements corresponding to some row permutation of L is smaller than the maximum by some positive integer m, then (41) implies $|\tilde{L}| = O(h^{-m})$, yielding a strongly coupled system. Moreover, if there is more than one permutation that maximizes the sum of the orders of the diagonal elements of L, then (39) cannot hold, since there is then at least one term in $|\tilde{L}|$ other than the product of the diagonal 1's which is $O(1)$ (the term(s) corresponding to the other permutation(s) that maximize the product of the orders of the diagonal elements of $|L|$). This implies uniqueness of the weakly coupled configuration.

3.3.3. Successive relaxation. Relaxation whereby the equations are relaxed in some prescribed order, with the variables updated immediately, yields a much more complicated relaxation operator. However, note the following.

THEOREM 5. *Theorem 4 holds for successive relaxation. Furthermore, every term in the rhs quantity for simultaneous relaxation appears in a term in the rhs quantity for successive relaxation, multiplied by one or more diagonal elements of $R - \lambda I$.*

Theorem 5 implies that, barring freak cancellation of terms which is discussed in §4, there is no qualitative difference in smoothing performance between simultaneous and successive relaxations. The sole exception is that in the (entirely moot) case where two or more μ_i's in a loop are approximately equal to zero, their multiplicity should be considered to be one. Indeed, only when *all* the smoothing factors vanish does this have any influence on the overall smoothing factor, and even then this influence is only on the rate at which the smoothing factor of the system tends to zero and not on whether or not it does.

The proof of Theorem 5 is quite lengthy and is only sketched here. In order to analyze successive relaxation, we first prescribe the order in which the equations are to be relaxed, then calculate the relaxation operator of each equation separately, and finally multiply the symbols (as is done in two-level analysis) to obtain the full successive relaxation symbol, which shall be denoted by $\hat{R}^s$ and its elements by $r^s_{i,j}$. The symbol of the relaxation operator for simultaneous relaxation analyzed above will be denoted by $\hat{R}^p$ and its elements by $r^p_{i,j}$. $\hat{R}^i$ will denote the symbol of the relaxation operator of equation i, $i = 1, \ldots k$. Its elements are given by

$$r^i_{j,m} = \begin{cases} r^p_{j,i} & \text{if } m = i, \\ 1 & \text{if } j = m \neq i, \\ 0 & \text{otherwise.} \end{cases}$$

That is, $\hat{R}^i$ acts as the identity matrix with respect to relaxation of all equations but the one relaxed, while the effect of relaxing equation i on the residuals of the other equations is the same as in simultaneous relaxation and is manifest in the corresponding column of the relaxation operator. $\hat{R}^s$ is defined by

$$\hat{R}^s = \hat{R}^k \cdot \hat{R}^{k-1}, \ldots, \hat{R}^1 . \tag{42}$$

Consider G^i, the graph corresponding to $\hat{R}^i$. It is obtained from G^p (the graph of $\hat{R}^p$) by eliminating all the edges whose tail-node is *not* node i, and adding single-edged loops of weight one at all nodes but i. By the definition of matrix multiplication it is straightforward to show that the elements of $\hat{R}^s$ are given by the following:

$r^s_{i,j} =$ the sum of the weights of all *consistently oriented* paths in G^p,

which lead from node j to node i,

where a consistently oriented path is defined as follows.

DEFINITION 5. Let $(n_0, n_1, \ldots, n_e)$ define a path of e edges from node n_0 to node n_e. Then the path is consistently oriented if for all $1 \leq j < e$, $n_{j-1} < n_j$, but $n_{e-1} \geq n_e$.

Now, since all terms in $r^s_{i,j}$ are paths from node j to node i, it follows from an analogous statement to Lemma 1 that the terms in rhs corresponding to the determinant of $\hat{R}^s$ are all products of loops (though not necessarily simple loops). Furthermore, since all simple loops can be constructed by concatenation of consistently oriented paths, all the terms in rhs of $\hat{R}^p$ must appear in rhs of $\hat{R}^s$, albeit multiplied by elements of the diagonal of $\hat{R}^s$ (which include the corresponding diagonal elements of $\hat{R}^p$ as terms). This outlines the proof of Theorem 5.

4. Discussion and conclusions. A systematic approach for developing smoothers for multigrid solvers of complicated systems of partial differential equations has been presented, and its power and straightforward implementation have been depicted by examples taken mostly from problems describing interesting physical phenomena.

The coupling analysis is derived from local mode analysis. Its main advantage is its straightforward and systematic implementation. Although general principles for relaxing systems have already been formulated by Brandt in [1], the present approach expands their scope, especially when used in conjunction with Brandt's approach. In particular, the coupling analysis offers a quantitative estimate of the strength of coupling. This, along with the structure of the coupling, which is clearly depicted by the coupling graph, suggests the methods of weakening coupling brought forth in §2.6.

The coupling analysis does not provide a precise quantitative prediction of the smoothing performance as does local mode analysis (although Rule 6 normally gives quite a narrow estimate). In particular, in some special cases for which the coupling analysis predicts strong coupling, the system may conceivably perform well under the usual simultaneous or successive relaxation of the equations, due to an "accidental" cancelling (or near-cancelling) of $O(1)$ terms in rhs of (33) (or of "perturbations" by the interpretation offered in the beginning of §2). But this may well be more of a help than a hindrance, since robustness considerations (with respect to size and variability of coefficients) will usually make such a relaxation scheme unattractive. Furthermore, once a strongly coupled loop has been diagnosed, it can be analyzed precisely by reverting to the actual symbols, rather than orders of magnitude. This reduces the problem to examining a scalar rather than a full matrix. It may then be possible to find a way to reduce the feedback sufficiently, by such means as under-relaxation or different relaxation schemes for the equations in the loop, or perhaps other means, as suggested by the feedback-system interpretation. This approach requires further research.

A potentially important notion, presented in §2.5, is the broadening of the scope of distributive relaxation to include transformed operators that are not triangular—only weakly coupled. Here and in other examples, the coupling analysis can be employed to determine the best way to use existing multigrid tools. The quick and simple application of the basic analysis, which is all that is required in the preliminary stage, allows even a trial and error search for a successful scheme, which may be useful in very complicated systems.

It is important to note that the performance predicted by the coupling analysis, as by the usual smoothing analysis, is asymptotic performance. A single cycle might exhibit reduced performance due to high-derivative off-diagonal terms in the operator. This can usually be overcome by using higher-order interpolation of the error-correction (and perhaps the solution itself in the full multigrid (FMG) algorithm) for some of the variables, or by an extra relaxation sweep for the corresponding equation(s), but, in practice, such measures are seldom necessary (see [6], for example). A discussion of required orders of interpolation appears in [1, §4.3] and [2].

Implementation of the coupling analysis can be facilitated by a small computer program that calculates the weight-array from the order-array and plots the coupling graph. Then, examination of even quite complicated systems becomes extremely fast, even on a trial and error basis. Indeed, a black-box type of search can be automatized, at least for the preliminary phase.

Acknowledgment. The author would like to thank Stephen F. McCormick for his valuable remarks.

REFERENCES

[1] A. Brandt, 1984 *Multigrid Guide with Applications to Fluid Dynamics*, Monograph, GMD-Studie 85, GMD-FIT, Postfach 1240, D-5205, St. Augustin 1, West Germany, 1985.

[2] ———, *Rigorous Quantitative Analysis of Multigrid*, Report, The Weizmann Institute of Science, Rehovot, Israel, 1990.

[3] A. Brandt and N. Dinar, *Multigrid solutions to elliptic flow problems*, in Numerical Methods for Partial Differential Equations 53, S. Parter, ed., 53 (1979), Academic Press; ICASE Report 79-15, NASA Langley Research Center, Hampton, VA, 1979.

[4] A. Brandt and I. Yavneh, *On multigrid solution of high-Reynolds incompressible entering flows*, J. Comput. Phys., 101 (1992), pp. 151–164.

[5] L. Lovasz and M. D. Plummer, *Matching theory*, Ann. Discrete Math., North Holland, Amsterdam, 1986.

[6] I. Yavneh and J. C. McWilliams, *Efficient Multigrid Solution of the Shallow Water Balance Equations*, J. Comput. Phys., submitted.

SIAM J. SCI. COMPUT.
Vol. 14, No. 6, pp. 1464–1486, November 1993

COMPUTING THE GENERALIZED SINGULAR VALUE DECOMPOSITION*

ZHAOJUN BAI† AND JAMES W. DEMMEL‡

Abstract. A variation of Paige's algorithm is presented for computing the generalized singular value decomposition (GSVD) of two matrices A and B. There are two innovations. The first is a new preprocessing step which reduces A and B to upper triangular forms satisfying certain rank conditions. The second is a new 2×2 triangular GSVD algorithm, which constitutes the inner loop of Paige's algorithm. Proofs of stability and high accuracy of the 2×2 GSVD algorithm are presented and are demonstrated using examples on which all previous algorithms fail.

Key words. generalized singular value decomposition, CS decomposition, matrix decomposition, Jacobi algorithm, Kogbetliantz algorithm

AMS subject classification. 65F30

CR subject classification. G1.3

1. Introduction. The purpose of this paper is to describe a variation of Paige's algorithm [28] for computing the following generalized singular value decomposition (GSVD) introduced by Van Loan [33] and Paige and Saunders [25]. This is also called the quotient singular value decomposition (QSVD) in [8].

THEOREM 1. *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times n}$ have* $\operatorname{rank}(A^T, B^T) = n$.[1] *Then there are orthogonal matrices U, V, and Q such that*

$$U^TAQ = \Sigma_1 R, \qquad V^TBQ = \Sigma_2 R, \tag{1.1}$$

where R is $n \times n$ upper triangular and nonsingular, and

$$\Sigma_1 = \begin{array}{c} \\ l \\ k \\ m-l-k \end{array} \left(\begin{array}{ccc} l & k & n-l-k \\ I_1 & & \\ & D_1 & \\ & & O_1 \end{array} \right), \tag{1.2}$$
$$\Sigma_2 = \begin{array}{c} \\ p-n+l \\ k \\ n-l-k \end{array} \left(\begin{array}{ccc} l & k & n-l-k \\ O_2 & & \\ & D_2 & \\ & & I_2 \end{array} \right),$$

$I_1 \in \mathbb{R}^{l\times l}$ and $I_2 \in \mathbb{R}^{(n-l-k)\times(n-l-k)}$ are identity matrices, $O_1 \in \mathbb{R}^{(m-l-k)\times(n-l-k)}$ and $O_2 \in \mathbb{R}^{(p-n+l)\times l}$ are zero matrices,

$$D_1 = \operatorname{diag}(\alpha_{l+1}, \ldots, \alpha_{l+k}), \qquad D_2 = \operatorname{diag}(\beta_{l+1}, \ldots, \beta_{l+k}), \tag{1.3}$$

*Received by the editors August 19, 1991; accepted for publication (in revised form) December 17, 1992. This work was supported in part by National Science Foundation grant ASC-9005933.

†Department of Mathematics, University of Kentucky, Lexington, Kentucky 40506. The work of this author was supported in part by National Science Foundation grant ASC-9102963.

‡Computer Science Division and Mathematics Department, University of California, Berkeley, California 94720. The work of this author was supported in part by Defense Advanced Research Projects Agency grant DAAL03-91-C-0047 via a subcontract to the University of Tennessee.

[1]The assumption that $\operatorname{rank}(A^T, B^T) = n$ is not essential but simplifies exposition.

$$(1.4) \qquad 1 > \alpha_{l+1} \geq \cdots \geq \alpha_{l+k} > 0, \quad 0 < \beta_{l+1} \leq \cdots \leq \beta_{l+k} < 1, \quad \alpha_i^2 + \beta_i^2 = 1.$$

The GSVD is a generalization of the singular value decomposition (SVD) in the sense that if B is the identity matrix, then the GSVD of A and B is the SVD of A. Moreover, if B is nonsingular, then the GSVD of A and B reduces to the SVD of AB^{-1}. If $(A^T, B^T)^T$ has orthonormal columns, then the GSVD of A and B is the CS decomposition [31]. The pairs (α_i, β_i) defined by the diagonal elements of Σ_1 and Σ_2 are called the *generalized singular value pairs* (GSV *pairs*). The quotient $\lambda_i = \alpha_i/\beta_i$ is called a *generalized singular value* (GSV). Note that the λ_i are the square roots of the eigenvalues of the symmetric pencil $A^TA - \lambda B^TB$.

The GSVD of two matrices A and B is a tool used in many applications, such as the Kronecker canonical form of a general matrix pencil [22], the linearly constrained least-squares problem [35], [5], the general Gauss–Markov linear model [27], [3], the generalized total least squares problem [21], and real time signal processing [30]. As a further generalization of the SVD, Ewerbring and Luk [13], Zha [36] proposed a generalized SVD for matrix triplets, and De Moor, Golub, and Zha [8], [9] have generalized the SVD into a factorization of any number of matrices. For all these applications and multimatrix generalization of the SVD, the development of a stable and efficient algorithm for computing the GSVD of two matrices is a basic problem.

Stewart [31] and Van Loan [34] proposed two algorithms for computing the GSVD. Their algorithms have two phases: The first phase is to compute the QR decomposition (or the SVD if necessary) of $(A^T, B^T)^T$. The second phase is to compute the CS decomposition. Paige's algorithm is a Jacobi–Kogbetliantz approach [28], which applies orthogonal transformations to A and B separately without the CS decomposition. It also has the following two phases.

Phase 1. Reduce matrices A and B to the following forms

$$(1.5) \qquad U^TAP = \begin{array}{c} \\ r \\ q \\ m-t \end{array} \begin{pmatrix} r & q & n-t \\ A_{11} & A_{12} & A_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad V^TBP = \begin{array}{c} \\ r \\ q \\ p-t \end{array} \begin{pmatrix} r & q & n-t \\ B_{11} & B_{12} & B_{13} \\ 0 & B_{22} & B_{23} \\ 0 & 0 & 0 \end{pmatrix},$$

where $m \times m$ matrix U and $p \times p$ matrix V are orthogonal, P is an $n \times n$ permutation matrix, $A_{11} \in \mathbb{R}^{r \times r}$ is nonsingular upper triangular, $B_{11} \in \mathbb{R}^{r \times r}$ is upper triangular, $t = r + q$, and if $q > 0$, $B_{22} \in \mathbb{R}^{q \times q}$ is nonsingular upper triangular.

Phase 2. Compute the GSVD of two $n \times n$ upper triangular matrices of forms (1.5) by a generalized Kogbetliantz algorithm.[2]

Phase 1 can be done first by the QR factorization with column pivoting [17] of matrix A and determine the rank r of A, meanwhile permuting the columns of matrix B in the

[2]We may need to add zero rows or columns to get square matrices. This is not essential but it simplifies the description.

same way, and then applying the QR factorization with column pivoting to the block of the last $p-r$ rows and $n-r$ columns of B and obtain the rank q of the block; this yields the forms (1.5) [4]. Phase 2 is iterative.

In this paper, we will present a variation of Paige's algorithm for computing the GSVD. There are two innovations. The first is as follows: In [28], it is assumed (without providing detail) that in (1.5) the nonzero part of V^TBP has full row rank. It is known that it is complicated to choose V to guarantee this condition and P may not be a permutation matrix. However, in the preprocessing step (1.5), we do not require this condition, and so we can simply use conventional QR factorization with column pivoting. Moreover, note that the GSVD is independent of column scaling of A and B. The forms (1.5) preserve this property.

The second innovation is a new 2×2 triangular GSVD algorithm, which constitutes the inner loop of Paige's algorithm. We will present proofs of stability and high accuracy of our method, and we will demonstrate it using examples on which all previous algorithms fail. Hereafter, we assume that A and B have been preprocessed to the upper trapezoidal forms (1.5).

The numerical technique developed in this paper can be extended to deal with the numerical computation of other closely related decompositions such as the CS decomposition and the product SVD of two matrices [20], [15]. We will not go into the details.

The rest of the paper is organized as follows. Section 2 reviews the Kogbetliantz algorithm for computing the SVD of a triangular matrix and Paige's generalization of the Kogbetliantz algorithm for computing the GSVD. Section 3 explores the inner loop of Paige's algorithm, which includes the GSVD of a 2×2 matrix in terms of exact and floating point arithmetic. In §4, we describe the overall algorithm. The last section reports the results of numerical experiments. In the Appendix, we include the 2×2 triangular SVD code of Demmel and Kahan, which has not been published in its entirety before, and which plays an important role in our algorithm.

2. Paige's GSVD algorithm.

To describe Paige's algorithm, we first review the Kogbetliantz algorithm [23] for computing the SVD of an upper triangular matrix A. Then we describe Paige's algorithm for computing the GSVD of A and B with B nonsingular. Finally, we discuss how to generalize the idea to the case where B is ill conditioned or singular.

2.1. Kogbetliantz algorithm for the SVD of a triangular matrix.

The Kogbetliantz algorithm [23] is a kind of Jacobi scheme. Assume that the kth transformation of the algorithm operates on the rows and columns i and j of A, let A_{ij} be the 2×2 submatrix subtended by rows and columns i and j of A. Let the rotation matrices $U_k = \mathrm{rot}(c_u, s_u)$ and $V_k = \mathrm{rot}(c_v, s_v)$ be chosen[3] so that

$$U_k^T A_{ij} V_k = \mathrm{diag}(\gamma_{ii}, \gamma_{jj})$$

is the SVD of A_{ij}, where $c_u = \cos\phi_k, s_u = \sin\phi_k$ and $c_v = \cos\psi_k, s_v = \sin\psi_k$. Let $\hat{U}_k$ and $\hat{V}_k$ be identity matrices with (i,i), (i,j), (j,i), and (j,j) elements replaced by the (1,1), (1,2), (2,1), and (2,2) elements of U_k and V_k, respectively. Then let

$$A_{k+1} = \hat{U}_k^T A_k \hat{V}_k,$$

where $A_0 = A$. After the first sweep through all the (i,j) in row cyclic order, an upper triangular matrix A will become lower triangular. The second sweep will restore upper

[3]Throughout this paper, we use $\mathrm{rot}(c,s)$ to denote the rotation matrix $\begin{pmatrix} c & s \\ -s & c\end{pmatrix}$.

triangular form and so on [20], [19]. There is a literature on the different sweep orders for sequential and parallel computations beside the conventional row and column order; for example [24].

Forsythe and Henrici [16] considered the convergence of the row cyclic Kogbetliantz algorithm. Fernando [14] proved a global convergence theorem under the assumption that one of the rotation angles $\{\phi_k, \psi_k\}$ at each (i,j) transformation lies in a closed interval $J \subset (-\pi/2, \pi/2)$, i.e.,

$$\phi_k \in J \quad \text{or} \quad \psi_k \in J, \qquad k = 1, 2, \ldots, . \tag{2.1}$$

This is the condition that our algorithm will satisfy. Furthermore, it has been proved that the cyclic Kogbetliantz algorithm ultimately converges quadratically [29], [2], [7].

2.2. The generalization of the Kogbetliantz algorithm for the GSVD. We begin by computing the GSVD of two upper triangular matrices A and B with B nonsingular. It is known that this is equivalent to computing the SVD of the triangular matrix $C = AB^{-1}$. Of course, it is unwise to form C explicitly. We note that a sweep of the Kogbetliantz algorithm applied to C will make it lower triangular. This means that there are orthogonal matrices U_1 and V_1 such that

$$U_1^T C V_1 = C_1, \tag{2.2}$$

where C_1 is lower triangular. Recasting (2.2) as $U_1^T A = C_1 V_1^T B$, we see that if we can determine an orthogonal matrix Q_1 satisfying

$$U_1^T A Q_1 = A_1, \qquad V_1^T B Q_1 = B_1,$$

where A_1 and B_1 are lower triangular, then $C_1 = A_1 B_1^{-1}$. This means that using a sweep of the Kogbetliantz algorithm on the upper triangular C to get the lower triangular C_1 is equivalent to the problem of finding orthogonal matrices U_1, V_1, and Q_1 so that $U_1^T A Q_1$ and $V_1^T B Q_1$ are lower triangular. Heath et al. [20], Paige [28], and Hari and Veselić [19] have shown that we may take advantage of the triangular structures of A and B and the ordering of sweeps to get the desired orthogonal transformations U_1, V_1, and Q_1 without forming AB^{-1} explicitly. Specifically, at the (i,j) transformation, the needed 2×2 submatrix C_{ij} of C is given by

$$C_{ij} = A_{ij} B_{ij}^{-1} = \begin{pmatrix} a_{ii} & a_{ij} \\ 0 & a_{jj} \end{pmatrix} \begin{pmatrix} b_{ii} & b_{ij} \\ 0 & b_{jj} \end{pmatrix}^{-1}, \tag{2.3}$$

where a_{ij} and b_{ij} are the elements subtended by the rows and columns i and j of the updated A and B, respectively. By using the SVD of C_{ij}: $U_{ij}^T C_{ij} V_{ij} = \mathrm{diag}(\tilde{c}_{ii}, \tilde{c}_{jj})$, we have

$$U_{ij}^T A_{ij} = \mathrm{diag}(\tilde{c}_{ii}, \tilde{c}_{jj}) V_{ij}^T B_{ij}.$$

This shows that the corresponding rows of $U_{ij}^T A_{ij}$ and $V_{ij}^T B_{ij}$ are parallel. Hence if we choose rotation Q_{ij} so that $V_{ij}^T B_{ij} Q_{ij}$ is lower triangular, then $U_{ij}^T A_{ij} Q_{ij}$ must also be lower triangular, which is just the GSVD of the 2×2 triangular matrices A_{ij} and B_{ij}. With this observation, we see that after completing a sweep in row order, the desired U_1, V_1, and Q_1 are the products $U_{12}U_{13} \cdots U_{n-1,n}$, $V_{12}V_{13} \cdots V_{n-1,n}$, and $Q_{12}Q_{13} \cdots Q_{n-1,n}$, respectively. By the end of the row cyclic sweep, we obtain lower triangular matrices

A_1 and B_1.[4] Then the next sweep consists of zeroing lower off-diagonal elements of $C_1 = A_1 B_1^{-1}$ in column order to return it to upper triangular form, and so on. Overall, we are actually carrying out the Kogbetliantz algorithm to diagonalize the implicitly defined matrix C. Upon convergence, this gives $U^T(AB^{-1})V = \Sigma$, a diagonal matrix. That is

$$U^T A Q = \Sigma \cdot V^T B Q,$$

i.e., the ith rows of $U^T AQ$ and $V^T BQ$ are parallel, which is the desired GSVD of A and B.

In general, if B is ill conditioned with respect to inversion or B is singular after phase 1, then using B_{ij}^{-1} is not recommended. Paige [28] suggests using

$$C_{ij} = A_{ij} \cdot \mathrm{adj}(B_{ij}) = \begin{pmatrix} a_{ii} & a_{ij} \\ 0 & a_{jj} \end{pmatrix} \begin{pmatrix} b_{jj} & -b_{ij} \\ 0 & b_{ii} \end{pmatrix} \tag{2.4}$$

instead of C_{ij} in (2.3), where $\mathrm{adj}(B_{ij})$ stands for the adjugate of B_{ij}. Since $B_{ij} \cdot \mathrm{adj}(B_{ij}) = \det(B_{ij})I$, it seems to be direct and natural to use $\mathrm{adj}(B_{ij})$ instead of B_{ij}^{-1}. The incorporation of (2.4) into the above procedure circumvents the numerical difficulties when B_{ij} is ill conditioned with respect to inversion or B_{ij} is singular. But it also introduces two questions. First, are there still rotation matrices U_{ij}, V_{ij}, and Q_{ij} such that $U_{ij}^T A_{ij} Q_{ij}$ and $V_{ij}^T B_{ij} Q_{ij}$ are the GSVD of 2×2 matrices A_{ij} and B_{ij}? Second, does the scheme converge to our required GSVD forms of A and B? The following section will address these questions.

3. The GSVD of 2×2 triangular matrices. As we see in §2, the kernel of computing the GSVD using a generalized Kogbetliantz algorithm is the computation of the GSVD of 2×2 matrices. In this section, we first discuss the computation of the 2×2 GSVD for different possible 2×2 matrices A_{ij} and B_{ij} in exact arithmetic, and then we will discuss the computation in the presence of floating point arithmetic.

3.1. The 2×2 GSVD in exact arithmetic. When A and B are processed to have upper trapezoidal forms (1.5), we see that at the (i,j) transformation, the 2×2 matrices A and B are of the forms[5]

$$A = \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{pmatrix}, \tag{3.1}$$

where $a_{11} \neq 0$, if A is nonzero. We have the following lemma.

LEMMA 1. *There exist* 2×2 *rotation matrices* U, V, *and* Q, *such that*

$$\tilde{A} = U^T AQ = \begin{pmatrix} \tilde{a}_{11} & 0 \\ \tilde{a}_{21} & \tilde{a}_{22} \end{pmatrix}, \qquad \tilde{B} = V^T BQ = \begin{pmatrix} \tilde{b}_{11} & 0 \\ \tilde{b}_{21} & \tilde{b}_{22} \end{pmatrix}$$

is the GSVD *of A and B. Moreover,*

[4]By incorporating Gentleman's suggested row and column permutations [28] after each transformation, we need only use an upper triangular array to carry out the computation. But, for clearer exposition, we will use the entire square array in this paper.

[5]For simplicity of exposition, we drop the subscript ij from the 2×2 triangular matrices A_{ij} and B_{ij}.

(a) $\tilde{a}_{11} \neq 0$ *if A is nonzero*,

(b) $\tilde{b}_{22} \neq 0$ *if both A and B are nonzero, except that*

(c) *if the first rows of A and B are parallel and the second rows are zero, then $U = V = I$, and Q can be chosen to zero the* (1,2) *entries of A and B simultaneously.*

Proof. The proof proceeds by considering all possible cases. If B is nonsingular, the lemma follows immediately by §2.2. If A or B is zero, the results are trivial. The remaining cases are for B singular but not zero. This includes the following three cases, where $C = A \cdot \mathrm{adj}(B)$:

(1) $$B = \begin{pmatrix} b_{11} & b_{12} \\ 0 & 0 \end{pmatrix} \quad \text{with} \quad b_{11} \neq 0.$$

In this case,

$$C = \begin{pmatrix} 0 & a_{12}b_{11} - a_{11}b_{12} \\ 0 & a_{22}b_{11} \end{pmatrix} \equiv \begin{pmatrix} 0 & c_{12} \\ 0 & c_{22} \end{pmatrix}.$$

If $c_{12} = 0$, i.e., the first row vectors of A and B are parallel, then if c_{22} is also equal to zero, $U = V = I$. Q_{ij} is chosen to zero (1,2) entry of A and must also zero the (1,2) entry of B, yielding the result (c). If $c_{22} \neq 0$, then both U and V are chosen as permutation matrices. Q is chosen to zero the (1,2) entry of $U^T A$.

If $c_{12} \neq 0$, then U is chosen to zero (2,2) entry of C and $V = \mathrm{rot}(0,1)$. $V^T B$ has second row nonzero. The lemma follows by choosing Q to zero (1,2) entry of $U^T A$.

(2) $$B = \begin{pmatrix} 0 & b_{12} \\ 0 & b_{22} \end{pmatrix} \quad \text{with} \quad b_{22} \neq 0.$$

Hence

$$C = a_{11} \begin{pmatrix} b_{22} & -b_{12} \\ 0 & 0 \end{pmatrix}.$$

Then $U = I$, and V is chosen to zero (1,2) entry of C, i.e., to zero the (1,2) entry of B. The lemma follows by choosing Q to zero (1,2) entry of A.

(3) $$B = \begin{pmatrix} 0 & b_{12} \\ 0 & 0 \end{pmatrix} \quad \text{with} \quad b_{12} \neq 0.$$

We see that

$$C = \begin{pmatrix} 0 & -a_{11}b_{12} \\ 0 & 0 \end{pmatrix}.$$

Then we can choose $U = I$, $V = \mathrm{rot}(0,1)$. Therefore the second row of $V^T B$ is nonzero. The lemma follows by choosing Q to zero (1,2) entry of $U^T A$. □

It has been shown by induction (see [28], [4]) that with the properties of Lemma 1, a sweep in row order with possible reordering takes the initial upper trapezoidal forms (1.5) of A and B into the forms

$$
A_1 = U_1^T A Q_1 = \begin{array}{c|cc} & r & n-r \\ \hline r & A_{11} & 0 \\ n-r & 0 & 0 \end{array},
$$

(3.2)

$$
B_1 = V_1^T B Q_1 = \begin{array}{c|cc} & r & n-r \\ \hline r & B_{11} & 0 \\ n-r & B_{21} & B_{22} \end{array},
$$

where A_{11}, B_{11}, and B_{22} are lower triangular, and A_{11}, B_{22} are nonsingular. B_{11} may be singular, but there must exist nonzero diagonal elements in the nonzero rows of B_{11}.

From (3.2), we see that at (i, j) transformation in column ordering, the 2×2 matrices A and B are lower triangular matrices, where if A is singular, then A is either the zero matrix or its second row is zero, and moreover, if $b_{22} = 0$, then $b_{21} = 0$. By a similar argument as in Lemma 1, we can show that there are 2×2 orthogonal matrices U, V, and Q such that $\tilde{A} = U^T AQ$ and $\tilde{B} = V^T BQ$ both are upper triangular and the GSVD of A and B. The proof of Lemma 1 suggests the following algorithm, where for brevity, we omit the part for lower triangular matrices.

ALGORITHM 1. (The 2×2 GSVD algorithm).
```
form $C = A \cdot \mathrm{adj}(B)$;
compute the SVD of C: $U^T CV = \mathrm{diag}(\sigma_1, \sigma_2)$;
form the products $G = U^T A$, $H = V^T B$;
if A is nonzero, then
    determine Q to zero out (1,2) entry of $G$;
else
    determine Q to zero out (1,2) entry of $H$;
end if
$\tilde{A} = GQ$;  $\tilde{B} = HQ$;  $\tilde{a}_{12} = 0$;  $\tilde{b}_{12} = 0$;
```

Again, from [28], [4], at the end of the second sweep, we have $A_2 = U_2^T A_1 Q_2$ and $B_2 = V_2^T B_1 Q_2$, such that

$$
(3.3)\quad A_2 = \begin{array}{c|ccc} & r_1 & r_2 & n-r \\ \hline r_1 & A_{11} & A_{12} & A_{13} \\ r_2 & 0 & A_{22} & A_{23} \\ n-r & 0 & 0 & 0 \end{array}, \qquad B_2 = \begin{array}{c|ccc} & r_1 & r_2 & n-r \\ \hline r_1 & 0 & 0 & 0 \\ r_2 & 0 & B_{22} & B_{23} \\ n-r & 0 & 0 & B_{33} \end{array},
$$

where A_{11}, A_{22}, B_{22}, and B_{33} are upper triangular matrices and nonsingular, $r_1 + r_2 = r$. Hence there is a unique $(n - r_1) \times (n - r_1)$ upper triangular matrix T such that

$$
\begin{pmatrix} A_{22} & A_{23} \\ 0 & 0 \end{pmatrix} = T \begin{pmatrix} B_{22} & B_{23} \\ 0 & B_{33} \end{pmatrix}.
$$

This implies that the rest of computation is essentially equivalent to computing the SVD of the implicitly defined matrix T. By the global convergence theory of the cyclic Kogbetliantz algorithm (see §2.1), we have

$$T \to \Sigma, \tag{3.4}$$

where Σ is a diagonal matrix, and the convergence is ultimately quadratic, provided the rotation angles of U and V obey (2.1). Equation (3.4) implies that there exists diagonal matrices Σ_1 and Σ_2 with $\Sigma_1^2 + \Sigma_2^2 = I$, and an upper triangular matrix R, such that

$$A_2 \to \Sigma_1 R \quad \text{and} \quad B_2 \to \Sigma_2 R,$$

which gives the desired GSVD of A and B.

3.2. The 2×2 GSVD in floating point arithmetic. In this section, we will use the usual model of floating point arithmetic: barring over/underflow, $\text{fl}(x \circ y) = (1+\delta)(x \circ y)$ where $\circ$ is one of the basic operations $\{+, -, \times, \div\}$, and $|\delta| \le \epsilon$ where ϵ is the machine round-off. This model eliminates machines like the Cray without guard digits, but with some effort all the results can be extended to these machines as well.

When using floating point arithmetic, round-off can cause the row vectors of $\tilde{A}$ and $\tilde{B}$ computed by Algorithm 1 not to be parallel. This means $\tilde{A}$ and $\tilde{B}$ are not the GSVD of the 2×2 matrices A and B, or, in short, the algorithm is not convergent. Another possibility is that the computation may not be backward stable, because the entries $\tilde{a}_{12}$ or $\tilde{b}_{12}$ ($\tilde{a}_{21}$ or $\tilde{b}_{21}$), which are explicitly set to zero by Algorithm 1, may be much larger than $O(\epsilon)\|A\|$ and $O(\epsilon)\|B\|$, respectively.[6] Thus, the algorithms in [28], [20], [4], which use the SVD of 2×2 triangular matrix to guarantee convergence, are potentially numerical unstable. On the other hand, to guarantee numerical stability, it is suggested in [18], [6] that after computing the SVD of the 2×2 triangular matrix C, one uses U (say) to form $G = U^T A$, then determines Q such that GQ is lower triangular and finally determines $\tilde{V}$ such that $\tilde{V}^T BQ$ is also lower triangular. However, in practice, $U^T C\tilde{V}$ might not be diagonal, which results in divergence. In §5, we will present numerical examples illustrating the failures of these schemes. In this section, we propose a new algorithm to overcome these shortcomings. We first discuss the two fundamental algorithmic building blocks: SLASV2 and SLARTG.

SLASV2 computes the SVD of a 2×2 upper triangular matrix

$$\begin{pmatrix} c_u & s_u \\ -s_u & c_u \end{pmatrix} \begin{pmatrix} f & g \\ 0 & h \end{pmatrix} \begin{pmatrix} c_v & -s_v \\ s_v & c_v \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}.$$

Barring over/underflow, SLASV2 computes all of c_u, s_u, c_v, s_v, σ_1, and σ_2 to nearly full machine precision. This algorithm was described briefly in [10], but not published in its entirety. For completeness, we include a listing of FORTRAN code in the Appendix and a statement and proof sketch of its error analysis. As discussed in [10], the high accuracy of SLASV2 is based on the fact that the algorithm uses formulas that only contain products, quotients, square roots, sums of terms of like sign, differences of computed quantities only when cancellation is impossible, and the difference $|f| - |h|$ of the input data, which, if cancellation occurs, is exact.[7]

[6]Throughout, $\|\cdot\|$ will denote the matrix 2-norm.

[7]This exact cancellation property, which is essential for the accuracy claim of SLASV2, requires a guard digit and so fails on machines like the Cray. On a Cray we retain backward stability of SLASV2, but lose forward stability. Since the proof uses forward stability of SLASV2 in an important way, it does not apply to Cray. However, there is a more complicated proof which does work on the Cray. The reader is invited to try to find it.

SLARTG(f, g, c, s, r) generates a rotation matrix $\mathrm{rot}(c, s)$ from f and g to zero g, i.e., $c = f/r$ and $s = g/r$, $r = \sqrt{f^2 + g^2}$, but this is subject to spurious over/underflow if we directly compute them from these formulas. A more robust way to compute c, s, and r can be found in [17]:

```
(handle $f = 0$ and $g = 0$ as special cases)
if $|f| > |g|$ then
    $t = g/f$; $tt = \sqrt{1+t^2}$; $c = 1/tt$; $s = t * c$; $r = f * tt$
else
    $t = f/g$; $tt = \sqrt{1+t^2}$; $s = 1/tt$; $c = t * s$; $r = g * tt$;
endif
```

The same techniques used to analyze SLASV2 in the Appendix can be straightforwardly used to show that the relative error in the computed c and s is bounded by 6ϵ.

Using SLARTG and SLASV2, we present a high-level description of an algorithm for computing the 2×2 GSVD. Later we will show that the proposed algorithm guarantees numerical stability and convergence. We will use the notation $|X| = (|x_{ij}|)$.

ALGORITHM GSVD22. Let A and B be 2×2 upper triangular matrices. The following algorithm computes the orthogonal matrices $U = \mathrm{rot}(c_u, s_u)$, $V = \mathrm{rot}(c_v, s_v)$, and $Q = \mathrm{rot}(c_q, s_q)$, such that

$$\tilde{A} = U^T A Q = \begin{pmatrix} \tilde{a}_{11} & 0 \\ \tilde{a}_{21} & \tilde{a}_{22} \end{pmatrix}, \qquad \tilde{B} = V^T B Q = \begin{pmatrix} \tilde{b}_{11} & 0 \\ \tilde{b}_{21} & \tilde{b}_{22} \end{pmatrix}$$

are the GSVD of A and B. For brevity, we omit the part for lower triangular matrices, which can be described similarly.

```
compute $C = A \cdot \mathrm{adj}(B)$;
use SLASV2 to compute the SVD of $C : U^T C V = \Sigma$;
compute $G = U^T A$;   $H = V^T B$;
compute $\hat{G} = |U|^T |A|$;   $\hat{H} = |V|^T |B|$;
/* The angles of U and V are chosen to satisfy the convergence condition (3.4). */
if $|c_u| \geq |s_u|$ or $|c_v| \geq |s_v|$ then
        /* Choose Q to zero out (1,2) entries of $U^T A$ and $V^T B$ */
        if $\hat{g}_{12}/(|g_{11}| + |g_{12}|) \leq \hat{h}_{12}/(|h_{11}| + |h_{12}|)$ then
            call SLARTG$(-g_{11}, g_{12}, c_q, s_q, r)$      /* Compute Q from $U^T A$ */
        else
            call SLARTG$(-h_{11}, h_{12}, c_q, s_q, r)$      /* Compute Q from $V^T B$ */
        end if
        $\tilde{A} = GQ$;  $\tilde{B} = HQ$;  $\tilde{a}_{12} = 0$;  $\tilde{b}_{12} = 0$.
else
        /* Choose Q to zero out (2,2) entries of $U^T A$ and $V^T B$ and then swap rows. */
        if $\hat{g}_{22}/(|g_{21}| + |g_{22}|) \leq \hat{h}_{22}/(|h_{21}| + |h_{22}|)$ then
            call SLARTG$(-g_{21}, g_{22}, c_q, s_q, r)$      /* Compute Q from $U^T A$ */
        else
            call SLARTG$(-h_{21}, h_{22}, c_q, s_q, r)$      /* Compute Q from $V^T B$ */
        end if
        $\tilde{A} = GQ$;  $\tilde{B} = HQ$;  $\tilde{a}_{22} = 0$;  $\tilde{b}_{22} = 0$.
        /* Swap, where $P = \mathrm{rot}(0,1)$ */
        $\tilde{A} \leftarrow P\tilde{A}$; $\tilde{B} \leftarrow P\tilde{B}$;
        $U \leftarrow UP$; $V \leftarrow VP$;
    end if
```

We now present a theorem about the stability and convergence of the above algorithm. Quantities with bars (like $\bar{C}$) denote computed quantities.

THEOREM 2. *The $\tilde{A}$ and $\tilde{B}$ computed by Algorithm* GSVD22 *have the following properties*:

(a) *Both are triangular*;

(b) *$\bar{U}^T\bar{C}\bar{V}$ is within* $132\epsilon\|\bar{C}\|$ *of being diagonal*;

(c) *The rows of $\bar{\tilde{A}}$ and $\bar{\tilde{B}}$ are within* $87\epsilon\|A\|$ *and* $87\epsilon\|B\|$, *respectively, of being parallel*;

(d) *They are computed stably, i.e., there exist δA and δB, where $\|\delta A\| \leq 377\epsilon\|A\|$ and $\|\delta B\| = 377\epsilon\|B\|$, and orthogonal U, V, and Q such that*

$$\bar{\tilde{A}} = U^T(A+\delta A)Q, \qquad \bar{\tilde{B}} = V^T(B+\delta B)Q,$$

Proof. We only prove a branch of the algorithm where Q is computed from U^TA and used to zero out the (1,2) entries of U^TA and V^TB; the proof for the other cases is similar. We will also leave some of the more tedious details of error analysis to the ambitious reader.

We first note the following facts about the algorithm.

Fact 1. $\bar{C} = (A+\delta A_1)\cdot \mathrm{adj}(B+\delta B_1)$, where δA_1 and δB_1 are small componentwise relative perturbations of A and B.

$$\bar{C} = \begin{bmatrix} a_{11}b_{22}(1+\epsilon_1) & -a_{11}b_{12}(1+2\epsilon_2)+a_{12}b_{11}(1+2\epsilon_3) \\ 0 & a_{22}b_{11}(1+\epsilon_4) \end{bmatrix}$$
$$= \begin{bmatrix} a'_{11}b_{22} & -a'_{11}b'_{12}+a'_{12}b'_{11} \\ 0 & a_{22}b'_{11} \end{bmatrix},$$

where $a'_{11} = a_{11}(1+\epsilon_1)$, $b'_{11} = b_{11}(1+\epsilon_4)$, $a'_{12} = a_{12}(1+2\epsilon_3)/(1+\epsilon_4)$, and $b'_{12} = b_{12}(1+2\epsilon_2)/(1+\epsilon_1)$. (The ϵ_i are independent quantities bounded in magnitude by the machine precision ϵ.) So there is at most a 3-ulp (unit in last place) perturbation in any entry, and also $\|\delta A_1\| \leq 4\epsilon\|A\|$ and $\|\delta B_1\| \leq 4\epsilon\|B\|$.

Fact 2. The computed $\bar{U}$ and $\bar{V}$ from SLASV2 satisfy $\bar{U} = U+\delta U$, $\bar{V} = V+\delta V$, where $U^T\bar{C}V$ is the exact SVD of $\bar{C}$, and δU (δV) is a small componentwise relative perturbation of U (V), bounded by 46.5ε in each component (see the proposition in the Appendix). This also implies $\|\delta U\| \leq \sqrt{2}\cdot 46.5\epsilon < 66\epsilon$ and $\|\delta V\| < 66\epsilon$.

Fact 3. The error in the $\bar{g}_{ij}$ ($\bar{h}_{ij}$) is bounded by $48.5\epsilon\bar{g}_{ij}$ ($48.5\epsilon\hat{h}_{ij}$). In the factor 48.5, two comes from the round-off in computing $\mathrm{fl}(U^TA)$ or $\mathrm{fl}(V^TB)$, and 46.5 comes from the errors in U and V.

Fact 4. Using simple geometry, one can show that changing f to $f+\delta f$ and g to $g+\delta g$ can change $c = f/\sqrt{f^2+g^2}$ and $s = g/\sqrt{f^2+g^2}$ to $c+\delta c$ and $s+\delta s$, respectively, where $\sqrt{\delta c^2+\delta s^2} \leq 2((\delta f^2+\delta g^2)/(f^2+g^2))^{1/2}$.

Fact 5. Subroutine SLARTG computes $c = f/\sqrt{f^2+g^2}$ and $s = g/\sqrt{f^2+g^2}$ with relative errors bounded by 6ϵ. This means the 2×2 matrix $\mathrm{rot}(c,s)$ has an error bounded in norm by $\sqrt{2}\cdot 6\epsilon < 9\epsilon$.

Fact 6. If X and Y are 2×2 matrices, then $\|\mathrm{fl}(X\cdot Y) - X\cdot Y\| \leq 4\cdot\epsilon\cdot\|X\|\cdot\|Y\|$.

We note that triangularity (a) holds by construction. We prove (b) as follows. Near diagonality of $\bar{U}^T\bar{C}\bar{V}$ holds by the high accuracy of $\bar{U}$ and $\bar{V}$:

$$\bar{U}^T\bar{C}\bar{V} = (U+\delta U)^T\bar{C}(V+\delta V) = U^T\bar{C}V + F_1,$$

where Fact 2 tells us that to first order in ϵ

$$\|F_1\| \le \|\delta U^T\bar{C}V\| + \|U^T\bar{C}\delta V^T\| \le 66\epsilon\|\bar{C}\| + 66\epsilon\|\bar{C}\| = 132\epsilon\|\bar{C}\|.$$

Next we prove assertion (c). The top rows of $\bar{\tilde{A}}$ and $\bar{\tilde{B}}$ are trivially parallel by construction (their second components are zero), so we only consider the bottom rows. We know by construction that the bottom rows of $U^T(A+\delta A_1)$ and $V^T(B+\delta B_1)$ are parallel. Thus the bottom rows of

$$\bar{G} = \text{fl}(\bar{U}^TA) = \bar{U}^TA + F_2 = (U^T+\delta U^T)(A+\delta A_1-\delta A_1) + F_2 = U^T(A+\delta A_1) + F_3$$

and $\bar{H} = V^T(B+\delta B_1)+F_4$ are within $\|F_3\| \le 74\epsilon\|A\|$ and $\|F_4\| \le 74\epsilon\|B\|$, respectively, of being parallel. Here we have used Facts 1, 2, and 6.

So for *any* $\bar{Q} = Q+\delta Q$ that is within 9ϵ in norm of an orthogonal matrix Q, the bottom rows of $\bar{\tilde{A}}$ and $\bar{\tilde{B}}$ are the same as the bottom rows of

$$\text{fl}(\bar{G}\bar{Q}) = \bar{G}\bar{Q} + F_5 = (U^T(A+\delta A_1)+F_3)(Q+\delta Q) + F_5 = U^T(A+\delta A_1)Q + F_6$$

and $\text{fl}(\bar{H}\bar{Q}) = V^T(B+\delta B_1)Q+F_7$, which are within $\|F_6\| \le 87\epsilon\|A\|$ and $\|F_7\| \le 87\epsilon\|B\|$ of being parallel; we have used our bounds on $\|F_3\|$ and $\|F_4\|$, and Facts 5 and 6. This proves assertion (c).

Let $\eta_a = \bar{\hat{g}}_{12}\epsilon/(|\bar{g}_{11}| + |\bar{g}_{12}|)$, and $\eta_b = \bar{\hat{h}}_{12}\epsilon/(|\bar{h}_{11}| + |\bar{h}_{12}|)$. Then η_a (η_b) is an approximate bound on relative error of Q if it is computed from U^TA (V^TB). In the branch of the algorithm we consider, $\eta_a \le \eta_b$, and the algorithm chooses to compute Q from U^TA. The remarkable fact is that even if $\epsilon \ll \eta_a$, so that the forward error in Q is large, the backward error in B is small.

To finally prove assertion (d), we need to show the (1,2) entry of $\text{fl}(\bar{H}\bar{Q})$, which is zeroed out to get $\tilde{B}$, is at most $286\epsilon\|B\|$. (Q is chosen to accurately zero out the (1,2) entry of $\text{fl}(\bar{G}\bar{Q})$.) Earlier we showed that $\bar{h}_{11} = h_{11}+74\epsilon_8\|B\|$ and $\bar{h}_{12} = h_{12}+74\epsilon_9\|B\|$, where h_{11} and h_{12} are the exact entries of $V^T(B+\delta B_1)$. Now write $\bar{c}_q = c_q+\delta c_q$ and $\bar{s}_q = s_q+\delta s_q$, where c_q and s_q are the exact cosine and sine computed from $U^T(A+\delta A_1)$. Then

$$\begin{aligned} |\text{fl}((\bar{H}\bar{Q})_{12})| &= |(h_{11}+76\epsilon_{10}\|B\|)(s_q+\delta s_q) + (h_{12}+76\epsilon_{11}\|B\|)(c_q+\delta c_q)| \\ &= |(h_{11}s_q + h_{12}c_q) + (h_{11}\delta s_q + h_{12}\delta c_q) + \sqrt{2}\cdot 76\epsilon_{12}\|B\|\,| \\ &\le |h_{11}|\,|\delta s_q| + |h_{12}|\,|\delta c_q| + 108\epsilon\|B\|. \end{aligned} \tag{3.5}$$

There are two cases, $\eta_a < \epsilon$ and $\eta_a \ge \epsilon$. In the first case, we will show that δs_q and δc_q are both bounded by 175ϵ, and so $|\text{fl}((\bar{H}\bar{Q})_{12})| \le 286\epsilon$. To see this, use Fact 3 to write

$$(48.5\epsilon\bar{\hat{g}}_{11})^2 + (48.5\epsilon\bar{\hat{g}}_{12})^2 = (48.5\epsilon\bar{g}_{11})^2 + (48.5\epsilon\bar{\hat{g}}_{12})^2 \le (48.5\epsilon)^2(\bar{g}_{11}^2 + (|\bar{g}_{11}| + |\bar{g}_{12}|)^2),$$

so by Facts 4 and 5, $\sqrt{|\delta s_q|^2 + |\delta c_q|^2}$ can be at most

$$9\epsilon + 2\cdot 48.5\epsilon\left(\frac{\bar{g}_{11}^2 + (|\bar{g}_{11}| + |\bar{\hat{g}}_{12}|^2)}{\bar{g}_{11}^2 + \bar{g}_{12}^2}\right)^{1/2} \le 178\epsilon\,.$$

Now we use this bound in inequality (3.5) to get $|\text{fl}((\bar{H}\bar{Q})_{12})| \le 286\epsilon$ as desired.

In the second case, we bound $\sqrt{|\delta s_q|^2 + |\delta c_q|^2}$ by

$$9\epsilon + 2\left(\frac{48.5^2((\epsilon\bar{g}_{11})^2 + (\eta_a(|\bar{g}_{11}| + |\bar{g}_{12}|))^2)}{\bar{g}_{11}^2 + \bar{g}_{12}^2}\right)^{1/2} \le 178\eta_a \,.$$

Plugging in to inequality (3.5) and using

$$|\bar{h}_{11}| + |\bar{h}_{12}| = \frac{\epsilon}{\eta_b}\bar{\hat{h}}_{12} \le \frac{\epsilon\|B\|}{\eta_b},$$

yields the upper bound

$$|\text{fl}((\bar{H}\bar{Q})_{12})| \le (|\bar{h}_{11}| + |\bar{h}_{12}|)\cdot 178\eta_a + 108\epsilon\|B\| \le \frac{\epsilon\|B\|}{\eta_b}\cdot 178\eta_a + 108\epsilon\|B\| = 286\epsilon\|B\|$$

as before, since $\eta_b \ge \eta_a$.

This means that we can write the final output

$$\bar{\bar{B}} = V^T(B+\delta B_1)Q + F_7 + F_8 = V^T(B+\delta B_1 + VF_7Q^T + VF_8Q^T)Q \equiv V^T(B+\delta B)Q,$$

where F_8 zeroes out the (1,2) entry of $\bar{\bar{B}}$ and leaves the others unchanged. We just showed $\|F_8\| \le 286\epsilon\|B\|$ and combining this with our earlier bounds of $\|F_7\| \le 87\epsilon\|B\|$ and $\|\delta B_1\| \le 4\epsilon\|B\|$ yields the final result $\|\delta B\| \le 377\epsilon\|B\|$. We can similarly show that $\bar{\bar{A}} = U^T(A + \delta A)Q$ with $\|\delta A\| \le 107\epsilon\|A\|$, using the fact that Q is computed to directly zero out the (1,2) entry of $\bar{\bar{A}}$. This completes the proof of assertion (d). $\square$

The constants in these error bounds could doubtless be decreased by a more detailed analysis.

4. Summary of the complete algorithm.

In this section, we present a high-level description of our version of Paige's algorithm for computing the GSVD of two upper triangular matrices A and B of the forms (1.5). Let κ be a user-chosen parameter specifying the maximum number of cycles the algorithm may perform (say, $\kappa = 20$). Let P_{ij} be the identity matrix with rows i and j interchanged.

```
ALGORITHM GSVD
    /* Initialization */
    cycle := 0;
    $\xi := r + q + 1$; /* r and q are defined in (1.5) */
    $U := I$; $V := I$; $Q := I$ if desired;
    /* Main loop */
    if nonconvergence and cycle $\le \kappa$ do
        cycle := cycle + 1;
        do $(i, j)$-loop
            /* 2 x 2 GSVD */
            Use GSVD22 to find $U_{ij}, V_{ij}, Q_{ij}$ from $a_{ii}$, $a_{ij}$, $a_{jj}$ and $b_{ii}$, $b_{ij}$, $b_{jj}$;
            /* Updating */
            $A := U_{ij}^T A Q_{ij}$;
            $B := V_{ij}^T B Q_{ij}$;
            $U := UU_{ij}$; $V := VV_{ij}$; $Q := QQ_{ij}$ if desired;
            /* reordering */
            if the $(j,j)$ entry of $B$ is nonzero, where $j > l$, then
                $A := AP_{\xi j}$;
                $B := P_{\xi j} B P_{\xi j}$;
                $V := VV_{\xi j}$;  $Q := QP_{\xi j}$ if desired;
                $\xi := \xi + 1$;
            end if
        end of $(i,j)$-loop
        convergence test if cycle is even.
    end if
    compute $\alpha_i$ and $\beta_i$.
```

The (i,j)-loop can be simply chosen as the standard cyclic pivot sequence. It is natural to use the parallelism (linear dependency) of the corresponding row vectors of A and B at the end of an even cycle as the stopping criterion of the iteration. To measure the parallelism of two k-vectors a and b to high accuracy and despite possible over/underflow, we propose the following scheme. First, compute the QR factorization of the $k \times 2$ matrix

$$\left(\frac{a}{\|a\|}, \frac{b}{\|b\|}\right):$$

$$Q^T \left(\frac{a}{\|a\|}, \frac{b}{\|b\|}\right) = \begin{pmatrix} \mu_{11} & \mu_{12} \\ 0 & \mu_{22} \\ 0 & 0 \end{pmatrix},$$

and then compute the singular values $\gamma_1 \geq \gamma_2 \geq 0$ of the 2×2 upper triangular (μ_{ij}). It is clear that

$$\operatorname{par}\left(\frac{a}{\|a\|}, \frac{b}{\|b\|}\right) \equiv \gamma_2$$

measures the parallelism of these two vectors. Vectors a and b are exactly parallel if and only if $\gamma_2 = 0$.

Using the above described scheme as the stopping criterion in Algorithm GSVD, let a_i and b_i be the ith row vectors of A and B, respectively, at the end of an even cycle. For a given tolerance value τ, we take

$$\text{error} = \sum_{i=1}^{n} \operatorname{par}\left(\frac{a_i}{\|a_i\|}, \frac{b_i}{\|b_i\|}\right) \leq n\tau.$$

This means that there are perturbations of size at most $n\tau\|a_i\|$ in row a_i and $n\tau\|b_i\|$ in row b_i that makes them exactly parallel. This means that after making these perturbations, there would exist scalars α_i and β_i such that

$$\beta_i a_i = \alpha_i b_i, \qquad i = 1, \ldots, n, \tag{4.1}$$

where α_i and β_i can be chosen so that $\alpha_i^2 + \beta_i^2 = 1$. From (4.1), it is seen that there is an upper triangular matrix R, such that

$$U^T AQ = \operatorname{diag}(\alpha_i)R, \qquad V^T BQ = \operatorname{diag}(\beta_i)R,$$

which is the desired GSVD of matrices A and B, where α_i and β_i are the GSV pairs.

5. Numerical experiments. The numerical experiments we discuss here first compare Algorithm GSVD22 with previous algorithms developed by Paige [28], Heath et al. [20], Bai [4], Hammarling [18], and Bojanczyk et al. [6]. Then we will evaluate Algorithm GSVD for different cases of random matrices A and B, measuring the backward stability, accuracy, average total number of sweeps, rate of convergence, elapsed time when computing GSV pairs only, or both GSV pairs, and transformation matrices.

All tests were performed using FORTRAN 77 on a SUN Sparc Station 1+. The arithmetic was IEEE standard double precision [1], with a machine precision of $\epsilon = 2^{-53} \approx 10^{-16}$ and over/underflow threshold $10^{\pm 307}$. We use $\tau = 10^{-14}$ as the stopping criterion.

5.1. Backward stability and accuracy. Before we proceed, it is appropriate to state what we mean by the *backward stability* and the *accuracy* of Algorithm GSVD. The backward stability is defined as follows: Let the computed orthogonal matrices be $\bar{U}$, $\bar{V}$, and $\bar{Q}$, the diagonal matrices be $\bar{\Sigma}_1$ and $\bar{\Sigma}_2$, and the upper triangular matrix be $\bar{R}$. Then the following conditions should be satisfied:

$$\|\bar{U}^T\bar{U} - I\|_F \approx \epsilon; \quad \|\bar{V}^T\bar{V} - I\|_F \approx \epsilon; \quad \|\bar{Q}^T\bar{Q} - I\|_F \approx \epsilon; \tag{5.1}$$

$$\|\bar{U}^T A\bar{Q} - \bar{\Sigma}_1\bar{R}\|_F \approx n\epsilon\|A\|_F; \qquad \|\bar{V}^T B\bar{Q} - \bar{\Sigma}_2\bar{R}\|_F \approx n\epsilon\|B\|_F, \tag{5.2}$$

where $\|\cdot\|_F$ is Frobenius norm. These assertions say that to within round-off error, the computed matrices $\bar{U}$, $\bar{V}$, and $\bar{Q}$ are orthogonal, and the rows of $\bar{U}^T A\bar{Q}$ and $\bar{V}^T B\bar{Q}$ are parallel.

The accuracy test of computed GSV pairs by Algorithm GSVD is based on the perturbation bound of the GSV pairs of Sun and Paige [32], [26], which says that: if $\mathrm{rank}(G) = \mathrm{rank}(\tilde{G}) = n$, where $G = (A^T, B^T)^T$ and $\tilde{G} = (\tilde{A}^T, \tilde{B}^T)^T = ((A+E)^T, (B+F)^T)^T$, and the GSV pairs (α_i, β_i) of A and B and $(\tilde{\alpha}_i, \tilde{\beta}_i)$ of $\tilde{A}$ and $\tilde{B}$ are ordered as in (1.4), respectively, then we have

$$\sqrt{\sum_{i=1}^{n}[(\alpha_i - \tilde{\alpha}_i)^2 + (\beta_i - \tilde{\beta}_i)^2]} \le \sqrt{2}\min\{\|G^\dagger\|_2, \|\tilde{G}^\dagger\|_2\}\left\|\begin{pmatrix} E \\ F \end{pmatrix}\right\|_F. \tag{5.3}$$

If we generate the matrices A and B with known GSV pairs, then the above perturbation bound can be used to measure the accuracy of the computed GSV pairs.

5.2. The numerical comparison of different 2 × 2 GSVD algorithms. Several versions have been proposed for computing the 2×2 GSVD. There are essentially two kinds of schemes.

Scheme 1. First compute the SVD of $C = A \cdot \mathrm{adj}(B)$: $U^TCV = \Sigma$, then form the product of $G = U^TA$ and $H = V^TB$, and finally compute Q from G such that the (1,2) or (2,1) entry of GQ is zero. Mathematically, it is known that the (1,2) or (2,1) entry of HQ is automatically zero. The algorithms proposed by Paige [28], Heath et al. [20], and Bai [4] fall in this category.

Scheme 2. First compute the SVD of $C = A \cdot \mathrm{adj}(B)$: $U^TCV = \Sigma$, form the product of $G = U^TA$, compute Q so such the (1,2) or (2,1) entry of GQ is zero, and finally compute V to zero out the (1,2) or (2,1) entry of BQ. The algorithms proposed by Hammarling [18] and Bojanczyk et al. [6] fall in this category.

To demonstrate the failure of the first kind of scheme, the following example shows that in finite precision, the (1,2) or (2,1) entry of the final B may be much larger than

$O(\epsilon)\|B\|$:

$$A = \begin{pmatrix} 2 & 0 \\ 1 & 10^{-8} \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}.$$

With the scheme described by Paige [28], Heath et al. [20], and Bai [4], for the computed $\bar{U}$, $\bar{V}$, and $\bar{Q}$, we have

$$\bar{U}^T A \bar{Q} = \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \end{pmatrix} = \begin{pmatrix} 0.70710677509009934E+00 & 0.21213203455917919E+01 \\ 0.00000000000000000E+00 & 0.28284271491319831E-07 \end{pmatrix},$$

$$\bar{V}^T B \bar{Q} = \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \end{pmatrix} = \begin{pmatrix} 0.31622776518779000E+00 & 0.94868330465141837E+00 \\ -0.33959487444334968E-08 & 0.31622776582710133E+01 \end{pmatrix}.$$

If we now set the (2,1) entry of $\bar{B} = \bar{V}^T B \bar{Q}$ to zero, the backward stability condition (5.2) is violated for matrix B, even though

$$\bar{U}^T C \bar{V} = \begin{pmatrix} 0.22360679640833827E+01 & 0.00000000000000000E+00 \\ 0.17888543605335784E-16 & 0.89442719636647925E-08 \end{pmatrix}.$$

To show how Scheme 2 can fail for the same example, using Hammarling's method [18], we have

$$\bar{U}^T A \bar{Q} = \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \end{pmatrix} = \begin{pmatrix} 0.70710677509009934E+00 & 0.21213203455917919E+01 \\ 0.00000000000000000E+00 & 0.28284271491319831E-07 \end{pmatrix},$$

$$\bar{V}^T B \bar{Q} = \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \end{pmatrix} = \begin{pmatrix} -0.31622776518778994E+00 & -0.94868327069193081E+00 \\ 0.11102230246251565E-15 & -0.31622776684588585E+01 \end{pmatrix}.$$

Thus the stability is achieved, but for the computed $\bar{U}$ and $\bar{V}$, we have

$$\bar{U}^T C \bar{V} = \begin{pmatrix} -0.22360679640833823E+01 & -0.24012983681642603E-07 \\ 0.78163392273857838E-16 & -0.89442719636647908E-08 \end{pmatrix},$$

which is not within $O(\epsilon)\|C\|$ of diagonal form. This means that the computed $\bar{A} = \bar{U}^T A \bar{Q}$ and $\bar{B} = \bar{V}^T B \bar{Q}$ are not the GSVD of A and B. In fact, $\text{par}((\bar{a}_1/\|\bar{A}\|), (\bar{b}_1/\|\bar{B}\|)) \approx \text{par}((\bar{a}_1/\|\bar{a}_1\|), (\bar{b}_1/\|\bar{b}_1\|)) \approx 7.59 \times 10^{-9}$.

But using Algorithm GSVD22 in §3.2, we have

$$\bar{U}^T A \bar{Q} = \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \end{pmatrix} = \begin{pmatrix} 0.70710677736817096E+00 & 0.21213203448324349E+01 \\ 0.30374288814267665E-16 & 0.28284271491319831E-07 \end{pmatrix},$$

$$\bar{V}^T B \bar{Q} = \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \end{pmatrix} = \begin{pmatrix} 0.31622776620657461E+00 & 0.94868330431182346E+00 \\ 0.00000000000000000E+00 & 0.31622776582710133E+01 \end{pmatrix},$$

and

$$\bar{U}^T C \bar{V} = \begin{pmatrix} 0.22360679640833827E+01 & 0.00000000000000000E+00 \\ 0.17888543605335784E-16 & 0.89442719636647925E-08 \end{pmatrix}.$$

Thus both stability and convergence conditions are satisfied, where $\mathrm{par}((\bar{a}_1/\|\bar{A}\|), (\bar{b}_1/\|\bar{B}\|)) \approx \mathrm{par}((\bar{a}_1/\|\bar{a}_1\|), (\bar{b}_1/\|\bar{b}_1\|)) \approx 7.02 \times 10^{-17}$.

Recently, Bojanczyk et al. [6] proposed a variation of Scheme 2, which we refer to as the BELV scheme. The BELV scheme was originally designed for treating a matrix-triple (A_1, A_2, A_3). It is easy to see that the 2×2 GSVD of two matrices is a special case when one of the matrices (say, A_3 is the identity). The BELV scheme does significantly improve Hammarling's method, but it still suffers from possible nonconvergence. For example, using the BELV scheme, we see that for the following 2×2 matrices

$$A = \begin{pmatrix} 100000 & 10000 \\ 0 & 0.0001 \end{pmatrix}; \quad B = \begin{pmatrix} 100000 & 10000.0000000001 \\ 0 & 0.003 \end{pmatrix},$$

the computed orthogonal matrices $\bar{U}$, $\bar{V}$, and $\bar{Q}$ by BELV scheme satisfy the stability conditions (5.1) and (5.2):

$$\bar{U}^T A \bar{Q} = \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \end{pmatrix} = \begin{pmatrix} 0.99503719020998935E-04 & -0.12189168086858831E-03 \\ 0.00000000000000000E+00 & 0.10049875621120891E+06 \end{pmatrix},$$

$$\bar{V}^T B \bar{Q} = \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \end{pmatrix} = \begin{pmatrix} 0.29851115706299699E-02 & -0.36499576550546638E-02 \\ 0.00000000000000000E+00 & 0.10049875621120886E+06 \end{pmatrix}.$$

However, the computed $\bar{U}$ and $\bar{V}$ do not diagonalize the matrix C:

$$\bar{U}^T C \bar{V} = \begin{pmatrix} 0.99999999999999964E+01 & -0.67590597725531451E-09 \\ 0.20277179317658482E-07 & 0.30000000000000023E+03 \end{pmatrix},$$

since the off-diagonal elements are much larger[8] than $O(\epsilon)\|C\| \approx 10^{-14}$, $\mathrm{par}((\bar{a}_1/\|\bar{a}_1\|), (\bar{b}_1/\|\bar{b}_1\|)) = 6.44 \times 10^{-4}$, and even $\mathrm{par}((\bar{a}_1/\|\bar{A}\|), (\bar{b}_1/\|\bar{B}\|)) = 1.43 \times 10^{-12}$, so that the first rows of $\bar{A}$ and $\bar{B}$ are not parallel. But Algorithm GSVD22 yields

$$\bar{U}^T A \bar{Q} = \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \end{pmatrix} = \begin{pmatrix} -0.10049875621120894E+06 & 0.00000000000000000E+00 \\ -0.12189168086858835E-03 & -0.99503719020998963E-04 \end{pmatrix},$$

$$\bar{V}^T B \bar{Q} = \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \end{pmatrix} = \begin{pmatrix} -0.10049875621120886E+06 & -0.18189894035458565E-11 \\ -0.36567504260576521E-02 & -0.29851115706299699E-02 \end{pmatrix},$$

and

$$\bar{U}^T C \bar{V} = \begin{pmatrix} 0.30000000000000028E+03 & 0.16940658945086007E-20 \\ 0.00000000000000000E+00 & 0.99999999999999982E+01 \end{pmatrix}$$

[8]In [6] it is proven that the off diagonal elements should be $O(\epsilon)\|A\|\,\|B\| \approx 10^{-6}$, which is attained. Since $\|C\| = \|A\,\mathrm{adj}(B)\| \ll \|A\|\,\|B\|$, our bound is much tighter.

with $\mathrm{par}((\bar{a}_1/\|\bar{A}\|),(\bar{b}_1/\|\bar{B}\|)) = \mathrm{par}((\bar{a}_1/\|\bar{a}_1\|),(\bar{b}_1/\|\bar{b}_1\|)) = 0$, and $\mathrm{par}((\bar{a}_2/\|\bar{A}\|),(\bar{b}_2/\|\bar{B}\|)) < \mathrm{par}((\bar{a}_2/\|\bar{a}_2\|),(\bar{b}_2/\|\bar{b}_2\|)) = 1.72 \times 10^{-16}$.

The above examples show that Algorithm GSVD22 is superior to all previous schemes.

5.3. Test matrix generation for testing backward stability. To test the backward stability of Algorithm GSVD, we used the LAPACK test matrix generation suite [11] to generate different types of upper triangular matrices A and B. The conditioning of a generated upper triangular matrix can be controlled by the following parameters:

`dist` specifies the type of probability distribution to be used to generate the random matrices:
= U: uniform distribution on (0, 1);
= S: uniform distribution on (-1, 1);
= N: normal distribution on (0, 1).

`cond` specifies the reciprocal of the condition number of generated matrix, `cond` ≥ 1.

`mode` describes how the singular values d_i of generated matrix are to be distributed:
= 1: sets $d_1 = 1$ and $d_i = 1/\mathtt{cond}, i = 2, \ldots, n$;
= 2: sets $d_i = 1, i = 1, \ldots, n-1$ and $d_n = 1/\mathtt{cond}$;
= 3: sets $d_i = \mathtt{cond}^{-(i-1)/(n-1)}, i = 1, \ldots, n$;
= 4: sets $d_i = 1 - \frac{i-1}{n-1}(1 - 1/\mathtt{cond}), i = 1, \ldots, n$;
= 5: sets d_i to random in (1/cond , 1), their logarithms are uniformly distributed;
= 6: sets d_i to random numbers from same distribution as the rest of the matrix.

We generated 12 separate classes of upper triangular matrices A and B according to different choices of parameters `dist`, `cond`, and `mode`, since this allows us to form different types of matrices to fairly test the behavior of the algorithm. The 12 classes are listed in Table 5.1. Thus classes 1 to 6 consist of well-conditioned matrices B, and the conditioning of matrix A is changed from well to ill conditioned. Classes 7 to 10 consist of well-conditioned matrices A and the conditioning of matrix B is changed from moderate to ill conditioned. Classes 11 and 12 consist of moderately conditioned matrices A and B.

TABLE 5.1
Test matrices.

	A			B		
class	dist	cond	mode	dist	cond	mode
1	U	10	6	U	10	6
2	U	10^2	2	S	10	6
3	U	10^5	1	N	10	5
4	S	10^8	3	S	10	6
5	S	10^{12}	4	U	10	5
6	S	10^{14}	4	N	10	6
7	N	10	6	N	10^5	1
8	N	10	6	U	10^8	2
9	N	10	6	S	10^{12}	2
10	S	10	6	N	10^{14}	4
11	S	10^5	4	N	10^5	4
12	S	10^3	3	N	10^4	4

5.4. Test results. We tested the above 12 classes of matrix pairs of dimension of $n = 5, 10, 20, 50$. In each class of dimension 5 we generated 401 matrix pairs, in each class of dimension 10 we generated 301 matrix pairs, in each class of dimension 20 we generated 201 matrix pairs, and in each class of dimension 50 we generated 101 matrix pairs. This makes a total of 12,048 test matrix pairs.

Table 5.2 illustrates the average number of double sweeps required to converge with the tolerance value $\tau = 10^{-14}$, where a double sweep consists of a sweep of row ordering and a sweep of column ordering. None of 12,048 test matrix pairs failed to converge. The observed largest number of double sweeps required to converge was five. The backward stability conditions (5.1) and (5.2) held throughout the test. The following quadratic convergence rate of the algorithm is typical of what we observed:

cycle	2	4	6	8
error $= \sum_{i=1}^{n} \mathrm{par}(a_i, b_i)$	1.5094	$1.0252 \cdot 10^{-2}$	$9.4356 \cdot 10^{-9}$	$6.4874 \cdot 10^{-16}$,

where A and B are 50×50 matrices, the condition numbers for both matrices are about 10^4.

TABLE 5.2
Average number of double sweeps.

Class		1	2	3	4	5	6	7	8	9	10	11	12
n	5	2.29	2.40	2.02	2.00	2.19	2.18	2.07	2.05	2.00	2.12	2.01	1.93
	10	3.00	3.01	2.99	2.01	3.01	3.00	2.97	3.01	2.00	2.99	3.00	2.00
	20	3.26	3.50	3.07	2.19	3.53	3.30	3.05	3.21	2.98	3.23	3.21	2.20
	50	4.00	4.01	3.99	3.00	4.00	4.00	3.89	4.01	3.00	4.00	4.00	3.00

5.5. Test matrix generation for testing accuracy. To test accuracy of Algorithm GSVD, we generated random matrices A and B with known GSV pairs. Specifically, let $\Sigma_1 = \mathrm{diag}(\alpha_i)$ and $\Sigma_2 = \mathrm{diag}(\beta_i)$ be the given GSV pairs. Then we generated random orthogonal matrices U, V, and Q uniformly distributed with respect to Haar measure, and a random upper triangular matrix R with specified smallest singular value, and finally formed

$$A = U\Sigma_1 RQ^T \quad \text{and} \quad B = V\Sigma_2 RQ^T. \tag{5.4}$$

Hence the GSV pairs of A and B are known to be (α_i, β_i).

In this way we can generate random test matrices having any distribution of the GSV pairs, and

$$\|G^\dagger\|_2^{-1} = \sigma_{\min}(G) = \sigma_{\min}(R).$$

Hence $\sigma_{\min}(R)$ (the smallest singular value) gives the conditioning of the designed test matrix pair. If $\bar{\alpha}_i$ and $\bar{\beta}_i$ are computed the GSV pairs by Algorithm GSVD, then the quantity

$$\Delta_1 \equiv \left\{ \sum_{i=1}^{n} [(\alpha_i - \bar{\alpha}_i)^2 + (\beta_i - \bar{\beta}_i)^2] \right\}^{1/2} \sigma_{\min}(R) \tag{5.5}$$

should be $O(\tau)$, where $\tau = 10^{-14}$ is our stopping criterion.

We designed six different distributions of the GSV pairs as illustrated in the second column of Table 5.3, where α_i and β_i are normalized so that $\alpha_i^2+\beta_i^2 = 1$ for $i = 1, \ldots, n$ if necessary, (U(0,1),U(0,1)) means that GSV pairs (α_i, β_i) comes from the normalization of a pair of random numbers from a uniform distribution on the interval (0,1). cond is the reciprocal of the smallest singular value of the matrix R in (5.4). Note that some of the distributions of GSV are well separated, some of them are highly clustered or multiple.

TABLE 5.3
Average double sweeps and accuracy of computed GSV pairs.

Type	α_i, β_i	$\sigma_{\min}(R)$	Double sweeps for different n				Δ_1
			5	10	20	40	
1	U(0,1), U(0,1)	10	2.28	3.02	3.53	4.02	$1.51 \cdot 10^{-14}$
		10^{-6}	2.02	3.00	3.23	4.00	$1.21 \cdot 10^{-15}$
		10^{-12}	2.07	3.04	3.45	4.21	$8.64 \cdot 10^{-15}$
2	$1/i^2, 1$	10	2.01	2.62	3.00	3.00	$2.56 \cdot 10^{-15}$
		10^{-6}	2.00	2.61	3.00	3.00	$2.65 \cdot 10^{-15}$
		10^{-12}	2.00	2.61	3.00	3.10	$9.99 \cdot 10^{-15}$
3	$i, 1$	10	2.48	3.06	3.99	4.06	$2.52 \cdot 10^{-14}$
		10^{-6}	2.07	3.01	3.99	4.02	$9.71 \cdot 10^{-15}$
		10^{-12}	2.75	3.38	4.01	4.53	$3.89 \cdot 10^{-16}$
4	$1 + \mathrm{mod}(i, n/4 + 1), 1$	10	1.03	2.03	3.00	4.00	$7.33 \cdot 10^{-14}$
		10^{-6}	1.00	2.26	3.04	4.02	$5.11 \cdot 10^{-15}$
		10^{-12}	1.08	3.51	3.50	4.59	$1.95 \cdot 10^{-15}$
5	$1 - \frac{i-1}{n-1}(1 - \frac{1}{\mathrm{cond}}), 1$	10	2.06	3.00	3.55	4.14	$3.29 \cdot 10^{-14}$
		10^{-6}	2.01	3.01	3.62	4.18	$4.23 \cdot 10^{-15}$
		10^{-12}	2.01	3.01	3.62	4.20	$6.05 \cdot 10^{-15}$
6	$1, \mathrm{cond}^{-(i-1)/(n-1)}$	10	2.28	3.00	3.28	4.00	$1.51 \cdot 10^{-14}$
		10^{-6}	2.00	2.77	3.00	3.17	$2.98 \cdot 10^{-16}$
		10^{-12}	2.00	2.00	3.00	3.20	$1.14 \cdot 10^{-15}$

5.6. Test results. We generated several categories of matrix pairs according to three parameters: the dimension n, the smallest singular value of R ($\sigma_{\min}(R)$), and the type of distribution of GSV. We first separated test matrices with three possible values of $\sigma_{\min}(R) = 1, 10^{-6}, 10^{-12}$, i.e., corresponding to well, moderately, and ill-conditioned GSVD problems. For each $\sigma_{\min}(R)$, we tested matrices of dimension $n = 5, 10, 20, 40$ with six different distributions of GSV pairs as shown in Table 5.1. This makes a total of $3 \times 4 \times 6 = 72$ different classes of matrices. In each class of dimension 5 we generated 301 matrices, in each class of dimension 10 we generated 201 matrices, in each class of dimension 20 we generated 101 matrices, and in each class of dimension 40 we generated 51 matrices, for a total of 10,772 test matrix pairs.

Table 5.3 illustrates the average number of double sweeps and accuracy of the algorithm for different size of matrices. The preprocessing orthogonal transformations of A and B to upper trapezoidal forms (1.5) are performed using LINPACK QR decomposition subroutine DQRDC [12]. In all tests, the backward stability conditions (5.1) and (5.2) are satisfied, so we do not report the details here. Given the backward stability, we can assume that the backward errors E of A and F of B satisfy $O(\|E\|, \|F\|) = O(10^{-14})$. For each type of GSV distribution, we let the conditioning (i.e., $\sigma_{\min}(R)$) of the GSVD problems vary from well to moderate to ill conditioned, as indicated in column 3 of Table 5.3. The numbers in columns 4 to 7 are the average numbers of double sweeps needed for convergence. The last column of the table is the largest value of Δ_1 computed from the formula (5.5). We see that all computed results are as accurate as predicted.

Finally, we briefly report timing results. The codes have not been polished intensively in order to reduce the execution time. Table 5.4 illustrates the required time for a 50×50 matrix pair A and B with 5 double sweeps to satisfy the stopping criterion.

Appendix. The SVD of 2×2 triangular matrix. In this Appendix, for the convenience of the reader, we include the 2×2 triangular SVD algorithm of Demmel and

TABLE 5.4
CPU timing of the GSVD *of two* 50 × 50 *matrices.*

Timing in seconds with $\tau = 10^{-14}$		
	without U, V, Q	with U, V, Q
preprocessing	0.28	1.11
iteration	13.11	20.99

Kahan. The algorithm was used in their high relative accuracy bidiagonal SVD algorithm [10], but the algorithm details were not presented there.

It is known that the singular values of the 2 × 2 upper triangular matrix $\left(\begin{smallmatrix} f & g \\ 0 & h \end{smallmatrix}\right)$ are the values of the unobvious expression $\frac{1}{2}|\sqrt{(f+h)^2+g^2} \pm \sqrt{(f-h)^2+g^2}|$, of which the bigger is γ_1 and the smaller is $\gamma_2 = |fh|/\gamma_1$. The right singular vector row $(-s_v, c_v)$ turns out to be parallel to $(f^2 - \gamma_1^2, fg)$. After computing a right singular vector, the corresponding left singular vector is determined by $(c_u, s_u) = (fc_v + gs_v, hs_v)/\gamma_1$. But computing the singular values/vectors directly from these expressions is unwise because round-off can destroy all relative accuracy, and they can suffer from over/underflow in the squared subexpressions even when the singular values/vectors are far from over/underflow thresholds. Demmel and Kahan have carefully reorganized the computation as described in the following so that barring over/underflow and assuming a guard digit in subtraction, all output quantities are correct to within a few ulps. In IEEE arithmetic [1], the code works correctly even if one matrix entry is infinite. Overflow is impossible unless the largest singular value itself overflows or is within a few ulps of overflow. (On machines with partial overflow, like the Cray, overflow may occur if the largest singular value is within a factor of two of overflow.) Underflow is harmless if underflow is gradual. Otherwise, results may correspond to a matrix modified by perturbations of size near the underflow threshold.

The error analysis of the main path of the code depends on the fact that all the operations except two are (i) multiplication and division, where the relative error of the result is at most 1 ulp larger than the sum of the relative errors of the inputs; (ii) addition of positive quantities, where the relative error of the result is at most 1 ulp larger than the maximum of the relative errors of the inputs; and (iii) square root, where the relative error of the result is at most 1 ulp more than half the relative error of the input.

There are also two subtractions in the main code path. The first subtracts original data D = FA-HA, and so has a 1 ulp error. In the second, T=2-L with $0 \le$ L ≤ 1, the relative error in T can only be 1 ulp larger than the relative error in L. These rules are sufficient to straightforwardly bound the error in the main code path, provided we ignore second order terms. There is another path corresponding to the case where the off-diagonal g is much larger than the other two matrix entries, which is analyzed much more easily. Summarizing all these considerations we can easily prove the following.

PROPOSITION 3. *Barring over/underflow, and assuming there is a guard digit in subtraction, the relative errors in the computed singular values are at most* 7 *ulps, and the relative errors in the computed singular vectors are at most* 46.5 *ulps in each component.*

The comments in the following code indicate the error bound in ulp of each computed quantity.

```
      SUBROUTINE SLASV2( F, G, H, SSMIN, SSMAX, SNR, CSR, SNL, CSL )
      REAL CSL, CSR, F, G, H, SNL, SNR, SSMAX, SSMIN
C
C     Computes singular value decomposition of 2 x 2 triangular matrix:
C     [ CSL SNL ] . [ F G ] . [ CSR -SNR ] = [ SSMAX 0 ]
C     [-SNL CSL ]   [ 0 H ]   [ SNR CSR ]    [ 0 SSMIN ]
C     Absolute value of SSMAX is larger singular value, Absolute value of SSMIN
C     is smaller singular value. Both CSR**2+SNR**2=1 and CSL**2+SNL**2=1.
C
```

```
C     .. Parameters ..
      REAL        ZERO, HALF, ONE, TWO, FOUR
      PARAMETER ( ZERO = 0.0, HALF = 0.5, ONE = 1.0, TWO = 2.0, FOUR = 4.0 )
C     .. Local Scalars ..
      LOGICAL  GASMAL, SWAP
      INTEGER  PMAX
      REAL        A, CLT, CRT, D, FA, FT, GA, GT, HA, HT, L, M,
      REAL        MM, R, S, SLT, SRT, T, TEMP, TSIGN, TT
C     .. Intrinsic Functions ..
      INTRINSICABS, SIGN, SQRT
C
      FT = F
      FA = ABS( FT )
      HT = H
      HA = ABS( H )
      PMAX = 1                                    /* PMAX points to maximum absolute entry of matrix */
      SWAP = ( HA.GT.FA )
      IF( SWAP ) THEN
          PMAX = 3
          TEMP = FT
          FT = HT
          HT = TEMP
          TEMP = FA
          FA = HA
          HA = TEMP
      END IF                                      /* Now FA .ge. HA */
      GT = G
      GA = ABS( GT )
      IF( GA.EQ.ZERO ) THEN                       /* Diagonal matrix */
          SSMIN = HA
          SSMAX = FA
          CLT = ONE
          CRT = ONE
          SLT = ZERO
          SRT = ZERO
      ELSE
          GASMAL = .TRUE.
          IF( GA.GT.FA ) THEN
              PMAX = 2
              IF( ABS( FA / GA ).LE.EPS ) THEN    /* Case of very large GA, EPS is machine epsilon */
                  GASMAL = .FALSE.
                  SSMAX = GA                      /* 1 ulp error */
                  IF( HA.GT.ONE ) THEN
                      SSMIN = FA / ( GA / HA )    /* 2 ulps error */
                  ELSE
                      SSMIN = ( FA / GA )*HA      /* 2 ulps error */
                  END IF
                  CLT = ONE                       /* 1 ulp error */
                  SLT = HT / GT                   /* 1 ulp error */
                  SRT = ONE                       /* 1 ulp error */
                  CRT = FT / GT                   /* 1 ulp error */
              END IF
          END IF
          IF( GASMAL ) THEN                       /* Normal case */
              D = FA - HA                         /* 1 ulp error */
              IF( D.EQ.FA ) THEN                  /* Copes with infinite F or H */
                  L = ONE                         /* 0 ulps error */
              ELSE
                  L = D / FA                      /* 2 ulps error */
              END IF                              /* Note that 0 ≤ L ≤ 1 */
              M = GT / FT                         /* 1 ulp error; Note that |M| ≤ 1/EPS */
              T = TWO - L                         /* 3 ulps error; Note that T ≥ 1 */
              MM = M*M                            /* 3 ulps error */
              TT = T*T                            /* 7 ulps error */
              S = SQRT( TT+MM )                   /* 5 ulps error; Note that 1 ≤ S ≤ 1 + 1/EPS */
              IF( L.EQ.ZERO ) THEN
                  R = ABS( M )                    /* 0 ulps error */
              ELSE
                  R = SQRT( L*L+MM )              /* 3.5 ulps error */
              END IF                              /* Note that 0 ≤ R ≤ 1 + 1/EPS */
              A = HALF*( S+R )                    /* 6 ulps error; Note that 1 ≤ A ≤ 1 + |M| */
              SSMIN = HA / A                      /* 7 ulps error */
              SSMAX = FA*A                        /* 7 ulps error */
              IF( MM.EQ.ZERO ) THEN               /* Note that M is very tiny */
                  IF( L.EQ.ZERO ) THEN
                      T = SIGN( TWO, FT )*SIGN( ONE, GT )     /* 0 ulps error */
                  ELSE
                      T = GT / SIGN( D, FT ) + M / T          /* 6 ulps error */
                  END IF
              ELSE
                  T = ( M / ( S+T )+M / ( R+L ) )*( ONE+A )   /* 17 ulps error */
              END IF
              L = SQRT( T*T+FOUR )                /* 18.5 ulps error */
              CRT = TWO / L                       /* 19.5 ulps error */
              SRT = T / L                         /* 36.5 ulps error */
              CLT = ( CRT+SRT*M ) / A             /* 46.5 ulps error */
              SLT = ( HT / FT )*SRT / A           /* 45.5 ulps error */
          END IF
      END IF
      IF( SWAP ) THEN
          CSL = SRT
          SNL = CRT
          CSR = SLT
          SNR = CLT
      ELSE
          CSL = CLT
          SNL = SLT
          CSR = CRT
```

```
         SNR = SRT
      END IF
C     Correct the signs of SSMAX and SSMIN
      IF( PMAX.EQ.1 ) TSIGN = SIGN( ONE, CSR )*SIGN( ONE, CSL )*SIGN( ONE, F )
      IF( PMAX.EQ.2 ) TSIGN = SIGN( ONE, SNR )*SIGN( ONE, CSL )*SIGN( ONE, G )
      IF( PMAX.EQ.3 ) TSIGN = SIGN( ONE, SNR )*SIGN( ONE, SNL )*SIGN( ONE, H )
      SSMAX = SIGN( SSMAX, TSIGN )
      SSMIN = SIGN( SSMIN, TSIGN*SIGN( ONE, F )*SIGN( ONE, H ) )
      RETURN
      END
```

Acknowledgment. The authors are grateful to S. Hammarling and W. Kahan for valuable comments. We would also like to express thanks to the referees whose comments led to improvement in the presentation.

REFERENCES

[1] *IEEE Standard for Binary Floating Point Arithmetic*, ANSI/IEEE, New York, Std 754-1985 Edition, 1985.

[2] Z. Bai, *Note on the quadratic convergence of Kogbetliantz algorithm for computing the singular value decomposition*, Linear Algebra Appl., 104 (1988), pp. 131–140.

[3] ———, *Numerical treatment of restricted Gauss–Markov linear model*, Comm. Statist., B17, 2 (1988), pp. 131–140.

[4] ———, *The direct* GSVD *algorithm and its parallel implementation*, Ph.D. thesis, Fudan University, China, 1987. Also available as Comput. Sci. TR-1901, Univ. of Maryland, College Park, MD, 1987.

[5] J. L. Barlow, *Error analysis and implementation aspects of deferred correction for equality constrained least squares problems*, SIAM J. Numer. Anal., 25 (1988), pp. 1340–1358.

[6] A. W. Bojanczyk, M. Ewerbring, F. T. Luk, and P. Van Dooren, *An accurate product* SVD *algorithm*, in SVD and Signal Processing, II, Algorithms, Analysis and Applications, R. J. Vaccaro, ed., Elsevier Sci. Publishers, North Holland, Amsterdam, 1991.

[7] J. P. Charlier and P. Van Dooren, *On Kogbetliantz's* SVD *algorithm in the presence of clusters*, Linear Algebra Appl., 95 (1987), pp. 136–160.

[8] B. L. R. De Moor and G. H. Golub, *Generalized singular value decompositions: A proposal for a standardized nomenclature*, Manuscript NA-89-05, Stanford Univ., Stanford, CA, 1989.

[9] B. L. R. De Moor and H. Zha, *A tree of generalizations of the ordinary singular value decomposition*, ESAT-SISTA Report 1989-21, Katholieke Universiteit Leuven, Belgium, 1989.

[10] J. Demmel and W. Kahan, *Accurate Singular Values of Bidiagonal Matrices*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 873–912.

[11] J. Demmel and A. Mckenney, LAPACK *working notes # 9, a test matrix generation suite*, MCS-P69-0389, Argonne National Lab, Argonne, IL, 1989.

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

[13] L. M. Ewerbring and F. T. Luk, *Canonical correlations and generalized* SVD: *Applications and New Algorithms*, J. Comput. Appl. Math., 27 (1989), pp. 37–52.

[14] K. V. Fernando, *Linear convergence of the row cyclic Jacobi and Kogbetliantz methods*, Numer. Math., 56 (1989), pp. 73–91.

[15] K. V. Fernando and S. J. Hammarling, *A product induced singular value decomposition for two matrices and balanced realization*, in Linear Algebra in Signals Systems and Control, B. N. Datta, C. R. Johnson, M. A. Kaashoek, R. J. Plemmons, and E. D. Sontag, eds. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 128–140.

[16] G. E. Forsythe and P. Henrici, *The cyclic Jacobi method for computing the principal values of a complex matrix*, Trans. Amer. Math. Soc., 94 (1960), pp. 1–23.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.

[18] S. J. Hammarling, private communications, 1989.

[19] V. Hari and K. Veselíc, *On Jacobi methods for singular value decomposition*, SIAM J. Sci. Statist. Comput., 6 (1987), pp. 741–754

[20] M. T. Heath, J. A. Laub, C. C. Paige, and R. C. Ward, *Computing the singular value decomposition of a product of two matrices*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1147–1159.

[21] S. V. Huffel and J. Vandewalle, *Analysis and properties of the generalized total least squares problem* $AX \approx B$ *when some or all columns in A are subject to error*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 294–315.

[22] B. K. ÅNGSTRÖM, *The generalized singular value decomposition and* $(A - \lambda B)$*-problem*, BIT, 24 (1984), pp. 568–583.
[23] E. G. KOGBETLIANTZ, *Solution of linear equations by diagonalization of coefficients matrix*, Quart. Appl. Math., 13 (1955), pp. 123–132.
[24] F. T. LUK AND H. T. PARK, *On parallel Jacobi orderings*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 18–26.
[25] C. C. PAIGE AND M. A. SAUNDERS, *Towards a generalized singular value decomposition*, SIAM J. Numer. Anal., 18 (1981), pp. 398–405.
[26] C. C. PAIGE, *A note on a result of Sun Ji-guang: sensitivity of the* CS *and* GSV *decomposition*, SIAM J. Numer. Anal., 21 (1984), pp. 186–191.
[27] ———, *The general linear model and generalized singular value decomposition*, Linear Algebra Appl., 70 (1985), pp. 269–284.
[28] ———, *Computing the generalized singular value decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1126–1146.
[29] C. C. PAIGE AND P. VAN DOOREN, *A note on the convergence of Kogbetliantz's iterative algorithm for obtaining the singular value decomposition*, Linear Algebra Appl., 77 (1986), pp. 301–313.
[30] J. M. SPEISER AND C. F. VAN LOAN, *Signal processing computations using the generalized singular value decomposition*, Proc. SPIE Vol. 495, Real Time Signal Processing VII, 1984, pp. 47–55.
[31] G. W. STEWART, *Computing the* CS*-decomposition of a partitioned orthonormal matrix*, Numer. Math., 40 (1982), pp. 297–306.
[32] J. SUN, *Perturbation analysis for the generalized singular value problem*, SIAM J. Numer. Anal., 20 (1983), pp. 611–625.
[33] C. F. VAN LOAN, *Generalizing the singular value decomposition*, SIAM J. Numer. Anal., 13 (1976), pp. 76–83.
[34] ———, *Computing the* CS *and the generalized singular value decomposition*, Numer. Math., 46 (1985), pp. 479–491.
[35] ———, *On the method of weighting for equality-constrained least-squares problems*, SIAM J. Numer. Anal., 22 (1985), pp. 851–864.
[36] H. ZHA, *A numerical algorithm for computing restricted singular value decomposition of matrix triplets*, Linear Algebra Appl., 168 (1992), pp. 1–26.

SIAM J. Sci. Comput.
Vol. 14, No. 6, pp. 1487–1503, November 1993

THE USE OF THE L-CURVE IN THE REGULARIZATION OF DISCRETE ILL-POSED PROBLEMS*

PER CHRISTIAN HANSEN† AND DIANNE PROST O'LEARY‡

Abstract. Regularization algorithms are often used to produce reasonable solutions to ill-posed problems. The L-curve is a plot—for all valid regularization parameters—of the size of the regularized solution versus the size of the corresponding residual. Two main results are established. First a unifying characterization of various regularization methods is given and it is shown that the measurement of "size" is dependent on the particular regularization method chosen. For example, the 2-norm is appropriate for Tikhonov regularization, but a 1-norm in the coordinate system of the singular value decomposition (SVD) is relevant to truncated SVD regularization. Second, a new method is proposed for choosing the regularization parameter based on the L-curve, and it is shown how this method can be implemented efficiently. The method is compared to generalized cross validation and this new method is shown to be more robust in the presence of correlated errors.

Key words. ill-posed problems, regularization, L-curve, parameter choice, generalized cross validation, discrepancy principle

AMS subject classifications. 65R30, 65F20

1. Introduction. In many applications such as spectroscopy [1], seismography [13], and medical imaging [11], data are gathered by convolution of a noisy signal with a detector. A linear model of this process leads to an integral equation of the first kind:

$$\int_0^1 k(s,t)\,x(t)\,dt = y_0(s) + e(s). \tag{1}$$

Here, $y_0(s) + e(s)$ is the measured signal, $y_0(s)$ is the true signal, $e(s)$ is the unknown noise, and the *kernel function* $k(s,t)$ is the instrument response function.

Since the measured signal is usually available only at a finite number of values of s, the continuous model (1) is replaced by a discrete linear model equation

$$Kx = y_0 + e \equiv y, \tag{2}$$

where K is a matrix of dimension $m \times n$ and we assume that $m \geq n$. In all but trivial deconvolution problems, the continuous problem is *ill posed* in the sense that small changes in the data can cause arbitrarily large changes in the solution, and this is reflected in ill conditioning of the matrix K of the discrete model, increasing as the dimension of the problem increases. Thus attempts to solve (2) directly yield solution vectors that are hopelessly contaminated with noise.

Hence some sort of *regularization* of the problem is required to filter out the influence of the noise. Well-known regularization methods are Tikhonov regularization and the truncated singular value decomposition (SVD). A common feature of these regularization methods is that they depend on some regularization parameter that controls how much filtering is introduced by the regularization. Often the key issue in connection with

*Received by the editors October 23, 1991; accepted for publication (in revised form) December 31, 1992.

†UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark (`unipch@wuli.uni-c.dk`). The work of this author was partially supported by a travel grant from the Reinholdt W. Jorck og Hustrus Fond.

‡Computer Science Department and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742 (`oleary@cs.umd.edu`). The work of this author was supported by the Air Force Office of Scientific Research grant AFOSR-87-0158.

these methods is to find a regularization parameter that gives a good balance, filtering out enough noise without losing too much information in the computed solution.

The purpose of this paper is to propose new methods for the choice of the regularization parameter through use of the the *L-curve*. The L-curve is a plot—for all valid regularization parameters—of the size of the regularized solution versus the size of the corresponding residual. It was used by Lawson and Hanson [10] and further studied by Hansen [9]. In this work we establish two main results. First we give a unifying characterization of various regularization methods and show that the measurement of "size" is dependent on the particular regularization method chosen; for example, the 2-norm is appropriate for Tikhonov regularization, but a 1-norm in the coordinate system of the SVD is relevant to truncated SVD regularization. Second, we propose a systematic a posteriori method for choosing the regularization parameter based on this L-curve and show how this method can be implemented efficiently. We compare the method to generalized cross validation and the discrepancy principle.

Our analysis differs from the "asymptotic theory of filtering" [6] where the problem size (m and n) goes to infinity, in that we consider problems where the problem size is typically *fixed*, e.g., by the particular measurement setup. Thus we are ignoring the very important questions of convergence of the estimates as the model converges to the continuous problem or as the error converges to zero.

We give a unified survey of regularization methods in §2 and of algorithms for choosing the regularization parameter in §3. We investigate various important properties of the L-curve in §4 and demonstrate how a good regularization parameter can actually be computed from the L-curve. In §5 we discuss several important computational aspects of our method, and, finally, in §6 we illustrate the new method by numerical examples.

2. Regularization methods. Practical methods for solving the discretized problem (2) must diminish the influence of noise. They differ only in the way that they determine the filtering function for the noise. We illustrate this by discussing several of these methods in a common framework, using the SVD of the matrix K. Let

$$K = \sum_{i=1}^{n} \sigma_i \, u_i \, v_i^T . \tag{3}$$

Here, the left and right *singular vectors* u_i and v_i are orthonormal, and the *singular values* σ_i are nonnegative and nonincreasing numbers, i.e., $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. Common for all discrete ill-posed problems is that the matrix K has a cluster of singular values at zero and that the size of this cluster increases when the dimension m or n is increased.

Using the SVD of K, it is straightforward to show that the ordinary least squares solution to (2), the one characterized by solving the unconstrained problem

$$\min_x \|Kx - y\|_2, \tag{4}$$

can be written as

$$x_{\mathrm{LSQ}} = \sum_{i=1}^{n} \frac{\alpha_i}{\sigma_i} \, v_i, \tag{5}$$

where $\alpha_i = u_i^T y$.

The trouble with using the least squares solution x_{LSQ} is that error in the directions corresponding to small singular values is greatly magnified and overwhelms the information contained in the directions corresponding to larger singular values. Any practical method must therefore incorporate filter factors f_i, changing the computed solution to

$$x_{\text{filtered}} = \sum_{i=1}^{n} f_i \frac{\alpha_i}{\sigma_i} v_i, \tag{6}$$

where usually we take $0 \le f_i \le 1$. If each filter factor is equal to one, we have the least squares solution x_{LSQ}. The filtered residual vector corresponding to x_{filtered} is

$$r_{\text{filtered}} = \sum_{i=1}^{n} (1 - f_i)\, \alpha_i u_i + r_\perp,$$

where $r_\perp$ is the least squares residual, i.e., the component of y orthogonal to the vectors $u_1, \ldots, u_n$. Regularization methods differ only in how they choose the filter factors.

Perhaps the best known regularization method is the one due to Tikhonov [16], which chooses the solution x_λ that solves the minimization problem

$$\min_x \{ \|Kx - y\|_2^2 + \lambda^2 \|x\|_2^2 \}. \tag{7}$$

Here, the parameter λ controls how much weight is given to minimization of $\|x\|_2$ relative to minimization of the residual norm. In some applications it is not appropriate to minimize the 2-norm of the solution, but rather a seminorm $\|L\,x\|_2$ where L typically is a discrete approximation to some derivative operator. However, Eldén [2] has shown that it is always possible to transform such problems into a form where the 2-norm is minimized.

Another regularizing method is *truncated* SVD [7], [17], where one simply truncates the summation in (5) at an upper limit $k < n$, before the small singular values start to dominate.

Certain iterative methods for solving the least squares problem (4) have minimization properties. The *conjugate gradient* family of methods minimizes the least squares function over an expanding sequence of subspaces

$$\mathcal{K}_k = \text{span}\{K^T y, (K^T K) K^T y, \ldots, (K^T K)^{k-1} K^T y\},$$

keeping $\|x\|_2$ as small as possible. One formulation particularly suited to ill-posed problems is the LSQR implementation of Paige and Saunders [12].

Another popular regularizing technique is the *maximum entropy* principle; see, for example, [15]. It is based on the idea that since y contains error, it is not reasonable to ask that x reproduce it exactly, but only that it approximate it within the expected value of the norm of the error in y. Among all of the x vectors that satisfy

$$\|Kx - y\|_2 \le \lambda,$$

the maximum entropy method chooses the vector whose entropy

$$s(x) = -\sum_{i=1}^{n} x_i \ln(x_i / x_i^0) - (x_i - x_i^0)$$

is the largest, where x^0 is a nonnegative initial approximation and the admissible points x are also nonnegative.

Like the least squares formulation (4), these regularization methods produce solutions that can be characterized as solutions to minimization problems. We can think of them as minimizing the size of the solution x subject to keeping the size of the residual $r = Kx - y$ less than some fixed value, or in a dual sense as minimizing the size of r subject to keeping the size of x less than some value $M(\lambda)$. The way that "size" is measured varies from method to method, but many of these methods are defined in terms of the norms induced by the bases of the SVD. Although the 2-norm is invariant with respect to the choice of an orthonormal basis, other norms do not share this property. We will denote the 1-norm in the SVD basis by $\|\cdot\|_{\underline{1}}$, defined by

$$\|x\|_{\underline{1}} = \|\beta\|_1 \quad \text{if} \quad x = \sum_{i=1}^{n} \beta_i v_i,$$

$$\|r\|_{\underline{1}} = \|\gamma\|_1 + \|r_\perp\|_1 \quad \text{if} \quad r = \sum_{i=1}^{n} \gamma_i u_i + r_\perp.$$

We make a similar definition for the p-norms in the SVD basis, $p = 3, \ldots, \infty$. Moreover, we denote by $M(\lambda)$ the norm of the solution vector for regularization parameter λ.

Using this notation it is easy to verify that the regularization methods mentioned above have the characterizations shown in Table 1. For instance, for the truncated SVD, we have the relations

$$\|x\|_{\underline{1}} = \sum_{i=1}^{n} f_i \left| \frac{\alpha_i}{\sigma_i} \right|,$$

$$\|r\|_{\underline{1}} = \sum_{i=1}^{n} (1 - f_i) |\alpha_i| + \|r_\perp\|_1.$$

For $M(\lambda) = 0$, $x = 0$ and all filter factors are zero. As M increases, we want to increase the size of x as little as possible for a given decrease in the size of r. When the size of r has been reduced by by δ, the size of x is

$$\sum_{i=1}^{n} \frac{|\delta_i|}{\sigma_i},$$

with

$$\sum_{i=1}^{n} |\delta_i| = \delta.$$

Thus the minimal size of x is achieved by finding the largest integer k so that

$$\sum_{i=1}^{k} |\alpha_i| \le \delta,$$

and then setting

$$\delta_i = \begin{cases} \alpha_i, & i = 1, \ldots, k, \\ (M - \sum_{i=1}^{k} |\alpha_i|)\operatorname{sgn}(\alpha_i), & i = k+1, \\ 0, & i = k+2, \ldots, n. \end{cases}$$

TABLE 1
Various regularization methods and their characterizations.

Method	Minimizes	Domain	Filter factors
Tikhonov	$\|r\|_2$	$\{x : \|x\|_2 \leq M(\lambda)\}$	$f_i = \frac{\sigma_i^2}{\lambda^2+\sigma_i^2}$
Truncated SVD	$\|r\|_{\underline{1}}$	$\{x : \|x\|_{\underline{1}} \leq M(\lambda)\}$	$f_i = 1, i = 1, \ldots, k(M)$, $f_{k(M)+1} = \frac{\sigma_{k(M)+1}}{\alpha_{k(M)+1}} \left(M - \sum_{i=1}^{k(M)} \frac{\|\alpha_i\|}{\sigma_i}\right)$ $\mathrm{f}_i = 0, i = k(M)+2, \ldots, n$
l_∞	$\|r\|_{\underline{\infty}}$	$\{x : \|x\|_{\underline{\infty}} \leq M(\lambda)\}$	$f_i = \min(1, \frac{M\sigma_i}{\|\alpha_i\|})$
LSQR	$\|r\|_2$	$\{x \in \mathcal{K}_k : \|x\|_2 \leq M(\lambda)\}$	No simple formula
Maximum entropy	$\|r\|_2$	$\{x \geq 0 : s(x) \geq M(\lambda)\}$	No simple formula

There is no *simple formula* for the filter factors for LSQR; but we will show in forthcoming work that they can be computed easily from intermediate quantities in the LSQR algorithm. For maximum entropy, we are not aware of an algorithm for computing the filter factors.

To unify notation, we will denote the regularization parameter by λ, even for methods such as truncated SVD and LSQR in which the regularization is determined by a discrete value k. We will denote the function minimized by a regularization method by $\rho(\lambda)$ and the norm or function associated with the regularized solution vector x by $\eta(\lambda)$. As an example, for Tikhonov regularization we have $\rho(\lambda) = \|K\,x - y\|_2$ and $\eta(\lambda) = \|x\|_2$.

The method l_∞ in the table is one of a family of methods, based on the l_p norms using the singular vector basis. For a comparable value of the regularization parameter, the l_∞ method produces a solution that is much "less smooth" than that of the truncated SVD or the Tikhonov method. The truncated SVD (l_1) solution has no components in directions corresponding to small singular values. The Tikhonov (l_2) solution has small components in these directions. The l_p ($p > 2$) solutions have larger components, and the l_∞ solution has components of size comparable to those in the directions corresponding to large singular values. We note that these methods can also be generalized to weighted l_p norms.

From this discussion we see that *the choice of regularization method is a choice of an appropriate pair of functions ρ and η. The proper choice of the regularization parameter is a matter of choosing the right cutoff for the filter factors f_i, i.e., the breakpoint in the singular value spectrum where one wants the damping to set in.* Algorithms for choosing the regularization parameter are still a subject of research. In the next section we survey two proposals for choosing the regularization parameter. Their shortcomings lead us to propose choosing the parameter based on the behavior of the L-curve, a plot of $\eta(\lambda)$ vs. $\rho(\lambda)$. The remainder of the paper is devoted to a discussion of the properties of the L-curve, numerical issues in using it to choose a regularization parameter, and examples of its performance compared with other methods.

3. Choosing the regularization parameter.

3. Choosing the regularization parameter. We survey the discrepancy principle and generalized cross validation, and then we propose the new method based on the L-curve.

3.1. The discrepancy principle. Perhaps the simplest rule is to choose the regularization parameter to set the residual norm equal to some upper bound for the norm $\|e\|_2$ of the errors in the right-hand side. In connection with discrete ill-posed problems this is called the *discrepancy principle* [5, §3.3]. There is also a generalized discrepancy principle that takes errors in the matrix K into account [9]. A major disadvantage of this method—apart from the fact that often a close bound on $\|e\|_2$ is not known—is the generally accepted fact that the discrepancy principle "oversmooths" the solution: i.e., it will choose the Tikhonov parameter λ too large, will drop too many singular values in the truncated SVD, or will halt LSQR at too early a stage. Thus we will not recover all the information actually present in the given right-hand side y.

3.2. Generalized cross-validation. A more promising rule is *generalized cross-validation* (GCV) [4], [18]. The basic idea in cross-validation is the following: if any data point y_i is left out and a solution $x_{\lambda,i}$ is computed to the reduced problem of dimension $(m-1)\times n$, then the estimate of y_i computed from $x_{\lambda,i}$ must be a good estimate. While ordinary cross-validation depends on the particular ordering of the data, generalized cross-validation is invariant to orthogonal transformation (including permutations) of the data vector y.

The GCV function to be minimized in this method is defined by

$$\mathcal{G}(\lambda) \equiv \frac{\|K\,x(\lambda) - y\|_2^2}{(\text{trace}(I - K\,K(\lambda)^I))^2},$$

where $K(\lambda)^I$ is any matrix that maps the right-hand side y onto the solution $x(\lambda)$, i.e., $x(\lambda) = K(\lambda)^I y$.

Although GCV works well for many problems, there are some situations in which GCV has difficulty finding a good regularization parameter. One difficulty is that the GCV function can have a very flat minimum and hence the minimum itself may be difficult to localize numerically. This is illustrated in [17].

Another difficulty is that GCV can sometimes mistake correlated noise for a signal. The underlying assumption when deriving GCV, cf. [4], [18], is that the errors in the right-hand side are normally distributed with zero mean and covariance matrix $\sigma^2 I$. We state from [18, p. 65] that GCV "is fairly robust against nonhomogenity of variance and non-Gaussian errors.... However, the method is quite likely to give unsatisfactory results if the errors are highly correlated." We illustrate this difficulty with a numerical example in §6.

3.3. The L-curve method. Another, more recent, alternative is to base the regularization parameter on the so-called *L-curve* [9]. The L-curve is a parametric plot of $(\rho(\lambda), \eta(\lambda))$, where $\eta(\lambda)$ and $\rho(\lambda)$ measure the size of the regularized solution and the corresponding residual [10]. The underlying idea is that a good method for choosing the regularization parameter for discrete ill-posed problems must incorporate information about the solution size in addition to using information about the residual size. This is indeed quite natural, because we are seeking a fair balance in keeping both of these values small. The L-curve has a distinct L-shaped corner located exactly where the solution x_λ changes in nature from being dominated by regularization errors (i.e., by oversmoothing) to being dominated by the errors in the right-hand side. Hence the corner of the L-curve corresponds to a good balance between minimization of the sizes, and the corresponding regularization parameter λ is a good one.

A feature of the L-curve that has not previously been considered is that the 2-norm is not always the appropriate measure of the size of the solution and residual vectors. The natural way to measure size is induced by the choice of the regularization method. Referring to Table 1, we conclude that the 2-norm is natural for Tikhonov regularization, for example, while the $l_{\underline{1}}$ norm should be used for the truncated SVD, since *that is the norm in which it is optimal.*

The idea of using the corner of the L-curve as a means for computing a good regularization parameter was originally proposed in [9], where it is also demonstrated that under certain assumptions that this criterion is indeed similar to both GCV and the discrepancy principle. Experiments confirm that whenever GCV finds a good regularization parameter, the corresponding solution is located at the corner of the L-curve.

The L-curve method for choosing the regularization parameter has advantages over GCV: computation of the corner is a well-defined numerical problem, and the method is rarely "fooled" by correlated errors. Even highly correlated errors will make the size of the solution grow once the regularization parameter λ becomes too small, thus producing a corner on the L-curve. We make these statements more precise in the next section.

4. Properties of the L-curve.

4.1. The shape of the curve. Many properties of the L-curve for Tikhonov regularization are investigated in [9]. In particular, it is shown that under certain assumptions the L-curve (ρ, η) for Tikhonov regularization has two characteristic parts, namely, a "flat" part where the regularized solution x_λ is dominated by regularization errors and an almost "vertical" part where x_λ is dominated by the errors. The three assumptions made in [9] are:

1. The discrete Picard condition is satisfied, i.e., the coefficients $|\alpha_i|$ on average decay to zero faster than the singular values σ_i.
2. The errors in the right-hand side are essentially "white noise."
3. The signal-to-noise ratio is reasonably large.

It was also shown in [9], under assumptions 1–3, that for any method whose filter factors behave quantitatively like those for Tikhonov regularization, its 2-norm L-curve (discrete or continuous) will be close to that of Tikhonov regularization.

It is, however, possible to show that the L-curve will *always* have an L-shaped appearance. Assume that the right-hand side has a component in each singular direction. (If not, reduce the problem dimension by dropping the corresponding component in the SVD.) The only other assumption we need to make is that the desired solution vector is bounded in size by some number $\bar{M}$ that is less than the size of the least squares solution (5). Such an assumption is realistic in all practical problems; perhaps all we know is that $\bar{M}$ is less than 10^{10}, but that is all we assume for now.

Consider first the L-curve (ρ^2, η^2) for Tikhonov regularization. We know that

$$\eta^2(\lambda) = \|x(\lambda)\|_2^2 = \sum_{i=1}^{n} \frac{\sigma_i^2 \alpha_i^2}{(\lambda^2 + \sigma_i^2)^2}$$

and

$$\rho^2(\lambda) = \|r(\lambda)\|_2^2 = \sum_{i=1}^{n} \frac{\lambda^4 \alpha_i^2}{(\lambda^2 + \sigma_i^2)^2} + \|r_\perp\|_2^2.$$

Thus

$$\frac{d(\eta^2(\lambda))}{d\lambda} = -4\lambda \sum_{i=1}^{n} \frac{\sigma_i^2 \alpha_i^2}{(\lambda^2 + \sigma_i^2)^3}, \qquad \frac{d(\rho^2(\lambda))}{d\lambda} = 4\lambda^3 \sum_{i=1}^{n} \frac{\sigma_i^2 \alpha_i^2}{(\lambda^2 + \sigma_i^2)^3}.$$

Therefore $d(\eta^2)/d(\rho^2) = -\lambda^{-2}$. Evaluation of the second derivative shows that the L-curve is convex and becomes steeper as the parameter λ approaches the smallest singular value.

The truncated SVD solutions yield a piecewise linear L-curve (ρ, η) using the l_1 norm measure of size. On the ith segment, the size of the residual changes by $|\alpha_i|$, while the size of the solution changes by $-|\alpha_i|/\sigma_i$. Thus the slope of the ith segment is $-1/\sigma_i$, and the curve becomes steeper as the size of the residual decreases.

It is easy to show that the l_∞ method has a similar property: the slope of each segment is again $-1/\sigma_i$ for some value of i.

Thus we have shown that for Tikhonov regularization, truncated SVD, and the l_∞ method, the L-curves become vertical as the size of the residual approaches its lower limit. Note that the slopes in each of these three cases are determined by K alone, independent of the right-hand side.

In forthcoming work with G. W. Stewart it is shown that the L-curve for LSQR has similar behavior if the right-hand-side coefficients decay sufficiently rapidly. The behavior of the curve for the maximum entropy criterion is a topic for future research.

Thus the L-curve basically consists of a vertical part for values of $\eta(\lambda)$ near the maximum value and an adjacent part with smaller slope. The more horizontal part corresponds to solutions where the regularization parameter is too large and the solution is dominated by regularization errors. The vertical part corresponds to solutions where the regularization parameter is too small and the solution is dominated by right-hand-side errors magnified by the division by small singular values. This behavior does not rely on any additional properties of the problem, e.g., statistical distribution of the errors, the discrete Picard condition, etc.

The idea of the L-curve criterion for choosing the regularization parameter is to choose a point on this curve that is at the "corner" of the vertical piece. Having an upper bound $\bar{M}$ on the size of x prevents us from being fooled by any other corners that the L-curve may have; in the absence of other information, we seek the leftmost corner consistent with the bound $\bar{M}$. The following are two ways of viewing the problem of corner location.

1. We could seek the point on the curve closest to the origin. The definition of "closest" can vary from method to method. For example, Tikhonov regularization measures distance as $\rho + \lambda^2\eta$.

2. We could choose the point on the L-curve where the curvature is maximum. The curvature is a purely geometrical quantity that is independent of transformations of the regularization parameter. We discuss implementation of this idea in §5.

The rationale behind using the corner to find a regularization parameter λ is that the corner corresponds to a solution in which there is a fair balance between the regularization and perturbation errors—because the corner separates the horizontal part of the curve from the more vertical part. This choice of λ may lead to a slightly underregularized solution because the influence of the perturbation errors must become apparent before the corner appears.

We stress numerically reliable methods but emphasize the fact that the L-curve picture gives a check on any method for locating the corner, as well as further insight into problem behavior, and that the user should not fail to look at it. There is good reason to use a set of routines that provide reliable numerical methods as well as good graphics, such as the Matlab-based code of Hansen [8].

4.2. Distinguishing signal from noise. In our experiments we have found that in many cases it is advantageous to consider the L-curve (ρ, η) in a log-log scale. There is strong intuitive justification for this. Since the singular values typically span several orders of magnitude, the behavior of the L-curve is more easily seen in such a log-log scale. In addition, the log-log scale emphasizes "flat" parts of the L-curve where the variation in either ρ or η is small compared to the variation in the other variable. These parts of the L-curve are often "squeezed" close to the axes in a lin-lin scale. Hence the log-log scale actually emphasizes the corner of the L-curve. One more advantage of the log-log scale is that particular scalings of the right-hand side and the solution simply shift the L-curve horizontally and vertically. Thus we do all of our computations related to curvature on $(\log\rho, \log\eta)$.

The log-log transformation has a theoretical justification as well. Consider the (ρ, η) curve for the truncated SVD algorithm. Recall that the ρ is the $l_{\underline{1}}$ norm of the residual, while η is the $l_{\underline{1}}$ norm of the solution vector, and the curve consists of the points produced by the truncated SVD algorithm for various numbers of retained singular values, $1 \leq k \leq n$. Using (5) we see that as k is increased by 1, the change in ρ is $|\alpha_k|$, while the change in η is $|\alpha_k/\sigma_k|$. Thus the slope of the kth segment of the piecewise linear interpolant is $1/\sigma_k$, independent of the right-hand side for the problem. Therefore, there is no hope of distinguishing signal from noise by examining properties of the L-curve in the lin-lin scale.

In a log-log scale, however, the slope of the kth segment is the *relative* change in η divided by the *relative* change in ρ, and these behave quite differently for signal and noise. A noiseless signal for which the discrete Picard condition is satisfied has the property that the sequences $|\alpha_k|$ and $|\alpha_k/\sigma_k|$ both approach zero. Thus the relative change in η approaches zero, while the relative change in ρ is finite and nonzero. Therefore, for a signal, the L-curve in log-log coordinates becomes flat as k is increased.

Pure noise gives a quite different L-curve in log-log scale. If we assume that the error components $|\alpha_k|$ are roughly a constant value ϵ, then $\eta_k \approx \epsilon/\sigma_k$ and the relative change in η_k is approximately σ_{k-1}/σ_k. The relative change in ρ_k is $1/(m-k)$, so the slope of the piecewise linear interpolant is $(m-k)\sigma_{k-1}/\sigma_k$. The L-curve for noise in log-log scale therefore has a steep slope as k increases, unlike the flat curve of the signal.

The same conclusion holds for the l_∞ regularization method. Suppose we increase the norm of x from γ to $\hat{\gamma}$. Then the norm of the residual changes from $\max_i |\beta_i| - |\gamma\sigma_i|$ to $\max_i |\beta_i| - |\hat{\gamma}\sigma_i|$. The slope of the L-curve in lin-lin coordinates is not strongly dependent on the right-hand-side coefficients β_i. The picture is different in log-log coordinates, though. The relative change in the norm of x is $(\hat{\gamma}-\gamma)/\gamma$, and the relative change in the norm of r is $(\hat{\gamma}-\gamma)\sigma_i/(|\beta_i| - |\gamma\sigma_i|)$ for some value of i. For pure signal, as γ increases, the value of i will be small: since $\alpha_i/\sigma_i \rightarrow 0$ by the discrete Picard condition, the components corresponding to small singular values are not changed by further increase in the norm of x. Thus the L-curve will have moderate slope. For pure noise, $i = n$, and the L-curve will be quite steep as γ increases.

The L-curves for pure signal and pure noise in Tikhonov regularization have similar characters: both curves are steep in lin-lin scale as $\lambda \rightarrow 0$, but only the noise curve is steep in log-log scale. This can be shown either by appealing to the closeness of the truncated SVD and the Tikhonov solutions and residuals when both are measured in the 2-norm, or by rather tedious computations with the 2-norm.

5. Numerical issues in locating the corner of the L-curve. Although the L-curve is easily defined and quite satisfying intuitively, computing the point of maximum curvature in a numerically reliable way is not as easy as it might seem. Below, we discuss three cases of increasing difficulty. Throughout this section we use the notation $(\hat{\rho}, \hat{\eta})$ for the chosen measures of the size of the residual vector and the size of the solution vector. In practical computation these would probably be taken to be the *log*s of the $\underline{1}$, 2, or p norms of these vectors, or some weighted versions of these norms.

5.1. The ideal situation: The L-curve defined by a smooth, computable formula. If the functions $\hat{\rho}$ and $\hat{\eta}$ are defined by some computable formulas, and if the L-curve is twice continuously differentiable, then it is straightforward to compute the curvature $\kappa(\lambda)$ of the L-curve by means of the formula

$$\kappa(\lambda) = \frac{\hat{\rho}'\hat{\eta}'' - \hat{\rho}''\hat{\eta}'}{((\hat{\rho}')^2 + (\hat{\eta}')^2)^{3/2}}. \tag{8}$$

Here, $'$ denotes differentiation with respect to the regularization parameter λ. Any one-dimensional optimization routine can be used to locate the value of λ that corresponds to maximum curvature.

This situation arises when using Tikhonov regularization on a problem for which the singular values of the matrix K are known. It is practical computationally, since the effort involved in such a minimization is much smaller than that for computing the SVD.

5.2. Lacking a smooth, computable function defining the L-curve. In many situations we are limited to knowing only a finite set of points on the L-curve. This is the case, for example, for the truncated SVD and LSQR algorithms, and in these and other cases the underlying curve is not differentiable. Thus the curvature (8) cannot be computed and in fact may fail to exist. The same may be the case for problems where the regularized solution results from some black box routine.

In a computational sense, the L-curve then consists of a number of discrete points corresponding to different values of the regularization parameter at which we have evaluated $\hat{\rho}$ and $\hat{\eta}$. In many cases, these points are clustered, giving the L-curve fine-grained details that are not relevant for our considerations. For example, if there is a cluster of small singular values σ_{i_1} through σ_{i_2} with right-hand-side coefficients even smaller, then the L-curve for the truncated SVD will have a cluster of points for values of k from i_1 to i_2. This situation does not occur for Tikhonov regularization because all the components in the solution come in gradually as the filter factors change from zero to one.

We must define a differentiable, smooth curve associated with the discrete points in such a way that fine-grained details are discarded while the overall shape of the L-curve is maintained; i.e., we want the approximating curve to achieve local averaging while retaining the overall shape of the curve. A reasonable approach is therefore to base the approximating smoothing curve on cubic splines. If we fit a pair of cubic splines to $\hat{\rho}(\lambda)$ and $\hat{\eta}(\lambda)$, or if we fit a cubic spline to $\hat{\eta}(\hat{\rho})$, then we have difficulty with approximating the corner well because dense knots are required here. This conflicts with the purpose of the fit, namely, to locate the corner.

Instead, we propose fitting a *cubic spline curve* to the discrete points of the L-curve. Such a curve has several favorable features in connection with our problem: it is twice differentiable, it can be differentiated in a numerically stable way, and it has local shape-preserving features [3]. Yet we must be careful not to approximate the fine-grained details of clusters of points too well. Since a cubic spline curve does not intrinsically have the desired local smoothing property, we propose the following two-step algorithm for computing a cubic spline-curve approximation to a discrete L-curve.

ALGORITHM FITCURVE
1. Perform a local smoothing of the L-curve points, in which each point is replaced by a new point obtained by fitting a low-degree polynomial to a few neighboring points.
2. Use the new smoothed points as control points for a cubic spline curve with knots $1, \ldots, N+4$, where N is the number of L-curve points.

Step 1 essentially controls the level of fine-grained details that are ignored. We have good experience with fitting a straight line in the least squares sense to five points centered at the point to be smoothed (a 1-norm fit may also work well, but is more difficult to compute). We illustrate the use of this algorithm in §6 .

In connection with using this algorithm as a stopping criterion for LSQR or any other iterative method, we stress that it is our belief that any sophisticated stopping rule for regularizing iterative methods (GCV, locating the point closest to the origin, finding the point of maximum curvature, etc.) must go a few iterations too far in order to determine the corner of the L-curve.

5.3. Limiting the number of L-curve points. In many cases, evaluating points on the L-curve is computationally very demanding and one would prefer to compute as few points as possible. For such problems, with differentiable as well as nondifferentiable L-curves, we need an algorithm that tries to locate the corner of the L-curve efficiently.

Assume that one knows a few points on each side of the corner. Then the ideas from the previous section can be used to derive an algorithm that seeks to compute a sequence of new regularized solutions whose associated points on the L-curve (hopefully) approach its corner. The algorithm is as follows.

```
ALGORITHM FINDCORNER
1. Start with a few points $(\hat{\rho}_i, \hat{\eta}_i)$ on each side of the corner.
2. Use the ideas in Algorithm FITCURVE to find an approximating three-
   dimensional cubic spline curve $S$ for the points $(\hat{\rho}_i, \hat{\eta}_i, \lambda_i)$, where $\lambda_i$ is the reg-
   ularization parameter that corresponds to $(\hat{\rho}_i, \hat{\eta}_i)$.
3. Let $S_2$ denote the first two coordinates of $S$, such that $S_2$ approximates the L-
   curve.
4. Compute the point on $S_2$ with maximum curvature, and find the corresponding
   $\lambda_0$ from the third coordinate of $S$.
5. Solve the regularization problem for $\lambda_0$ and add the new point $(\hat{\rho}(\lambda_0), \hat{\eta}(\lambda_0))$ to
   the L-curve.
6. Repeat from Step 2 until convergence.
```

In step 2, it is necessary to introduce λ_i as a third coordinate of S because we need to associate a regularization parameter with every point on S_2. A two-dimensional spline curve with λ_i as knots does not provide this feature. We stress again that it is not suitable to fit individual splines to $\hat{\rho}_i$ and $\hat{\eta}_i$.

Initial points for step 1 can be generated by choosing very "large" and very "small" regularization parameters, for example, λ equal to σ_1, $\frac{1}{10}\sigma_1$, $10\sigma_n$, and σ_n. Since these initial points may be far from the corner, we found it convenient to introduce an artificial temporary point $(\min_i(\hat{\rho}_i), \min_i(\hat{\eta}_i))$ between the points corresponding to "large" and "small" λ. This temporary point is *replaced* by the first L-curve point $(\hat{\rho}(\lambda_0), \hat{\eta}(\lambda_0))$ computed in the first iteration.

6. Numerical examples. In this section we illustrate the theory from the previous sections with numerical examples. We consider a first-kind Fredholm integral equation which is a one-dimensional model problem in image reconstruction from [14]. In this model, the unknown function x is the original signal, the kernel function $k(s,t)$ is the point spread function of an infinitely long slit, and the right-hand side y is the measured signal: i.e., y consists of the original signal x integrated with $k(s,t)$ plus additional noise e. The kernel is given by

$$(9)\quad k(s,t) = (\cos s + \cos t)\left(\frac{\sin u}{u}\right)^2, \quad u = \pi(\sin s + \sin t), \quad s,t \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right].$$

Discretization is performed by means of simple collocation with delta functions as basis functions. Hence the vectors x and y are simply samples of the underlying functions. Throughout, the order of the matrix K is $m = n = 64$.

We consider two different right-hand sides, both generated by multiplying the matrix K times the corresponding true solution vector x. The first right-hand side, y_1, satisfies the discrete Picard condition; i.e., the Fourier coefficients $\alpha_i = u_i^T y_1$ decay to zero faster than the singular values σ_i, so the solution coefficients α_i/σ_i also decay to zero. This right-hand side y_1 corresponds to a solution x_1 with two "humps" given by

$$(10)\qquad x_1(t) = 2\,\exp(-6\,(t-0.8)^2) + \exp(-2\,(t+0.5)^2).$$

The second right-hand side y_2 only marginally satisfies the discrete Picard condition: the right-hand-side coefficients are artificially generated so that all the solution coefficients are of approximately the same size. This problem is harder to solve numerically than the first one. The norms of the two right-hand sides are $\|y_1\|_2 = 18.6$ and $\|y_2\|_2 = 19.3$.

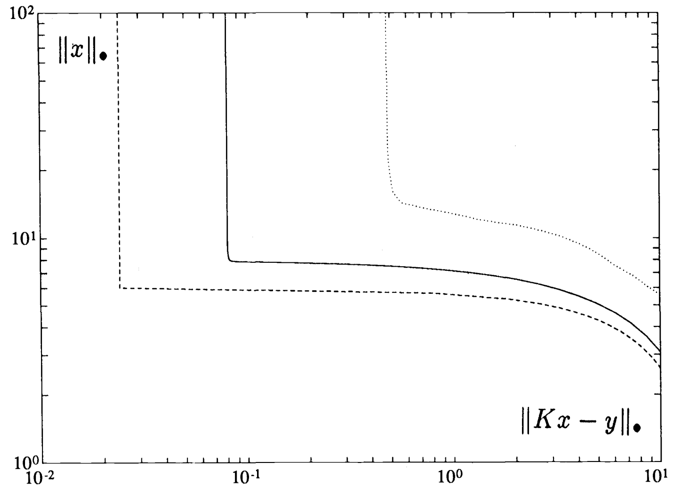
To each of these right-hand sides we add perturbation error consisting of normally distributed numbers with zero mean and standard deviation 10^{-2} so that $\|e\|_2 \approx 8\cdot 10^{-2}$.

The L-curves associated with truncated SVD, Tikhonov regularization, and the ℓ_∞ methods (see §2) are shown in Fig. 1. In lin-lin scale the truncated SVD and ℓ_∞ curves are piecewise linear, but these segments appear curved in the log-log scale. For both model problems and all three methods, the L-shaped appearance of the curves is very distinct. In particular, we notice the flat parts of the curves, corresponding to domination by the regularization error, and the vertical parts, corresponding to domination by perturbation errors. We also notice that even though the discrete Picard condition is barely satisfied for the second right-hand side y_2, the corresponding L-curves still have a distinct flat part.

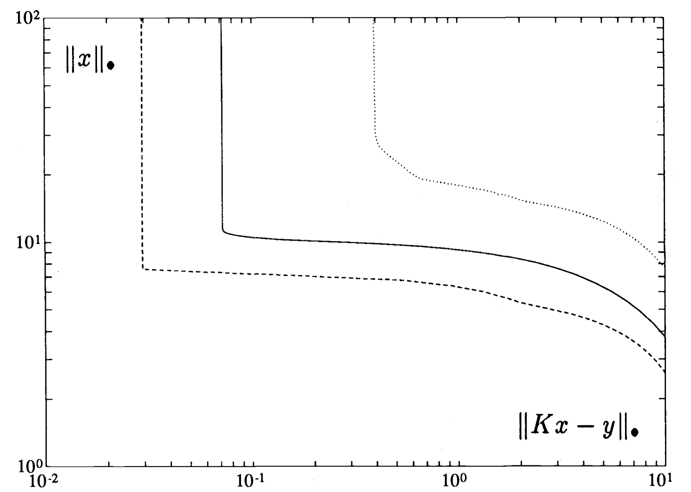
The rounded corner on the L-curve for truncated SVD shows the need for a rigorous definition of the "corner" of the L-curve—but in fact the other L-curves also have a rounded "corner" on a finer scale.

For the first model problem and for Tikhonov regularization, let us now consider the two types of errors in the regularized solution $x(\lambda)$. To this end, let $\hat{x}(\lambda)$ denote the part of $x(\lambda)$ solely from the unperturbed part of the right-hand side, such that $x(\lambda) - \hat{x}(\lambda)$ is the perturbation component of $x(\lambda)$. We want to compare the regularization error $\|x_1 - \hat{x}(\lambda)\|_2$ with the perturbation error $\|\hat{x}(\lambda) - x(\lambda)\|_2$. This is done in the left part of Fig. 2. Obviously, the regularization error increases with λ while the perturbation error decreases. The regularization parameter λ for which the two error types are identical can be characterized as the optimal λ.

The two vertical lines in Fig. 2 represent the regularization parameters chosen by means of the L-curve criterion (dashed-dotted line) and GCV (dotted line). The right part of the figure shows the corresponding GCV function and its minimum. Two typical situations are shown. In the upper part both GCV and the L-curve criterion yield approximately the same regularization parameter. In the bottom part of the figure the GCV function is quite flat, and the regularization parameter is a factor 10 too small. Thus Fig. 2 illustrates the major difficulty with the GCV method, namely, that the minimum is not always so well-defined.



(a)



(b)

FIG. 1. *The L-curves for model problems* (a) *one and* (b) *two for the the* ℓ_∞ *method* (*dashed line*), *Tikhonov regularization* (*solid line*), *and truncated SVD* (*dotted line*).

Let us now consider the robustness of the two competing methods in more detail. To do this, we compute the relative error in the regularized solution for both model problems:

$$\frac{\|x_i - x_\lambda\|_2}{\|x_i\|_2}, \qquad i = 1, 2,$$

computing the regularization parameter λ by both the L-curve criterion and by GCV. We used a broad range of error levels: $\|e\|_2/\|y\|_2 = 10^{-j}$, $j = 1, 2, \ldots, 8$, and for each error level we generated 25 error vectors. Thus for each model problem we solved 200 regularization problems. The results are shown in Figs. 3 and 4 as histograms with a logarithmic abscissa axis. The L-curve criterion rarely fails to compute a satisfactory

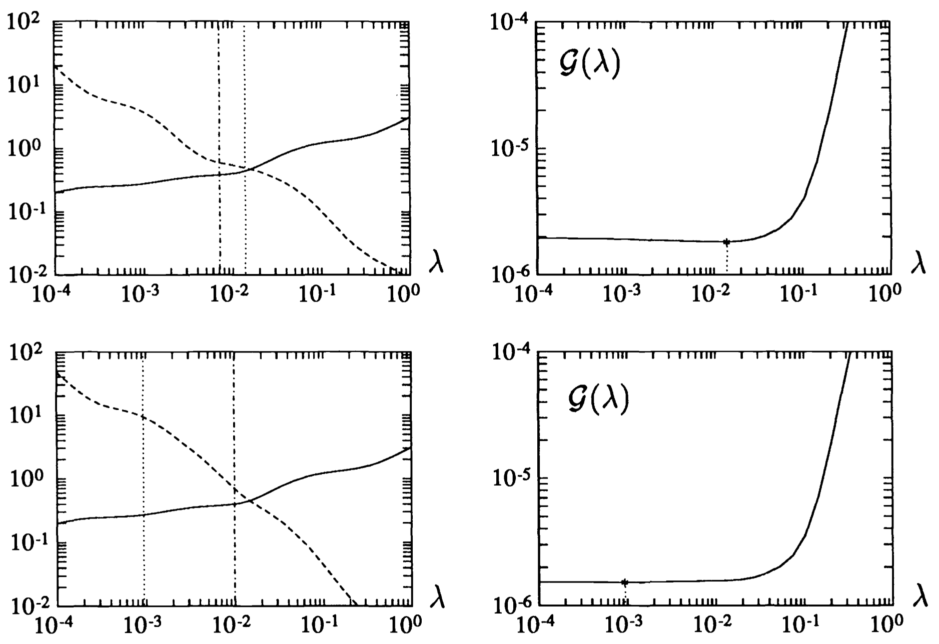


FIG. 2. *The left part shows the regularization error (solid line) and the perturbation error (dashed line) for model problem one, Tikhonov regularization, and two different random perturbations. The vertical lines represent the regularization parameters computed by means of the L-curve criterion (dashed-dotted line) and the GCV method (dotted line). The right part shows the corresponding GCV functions and their minima.*

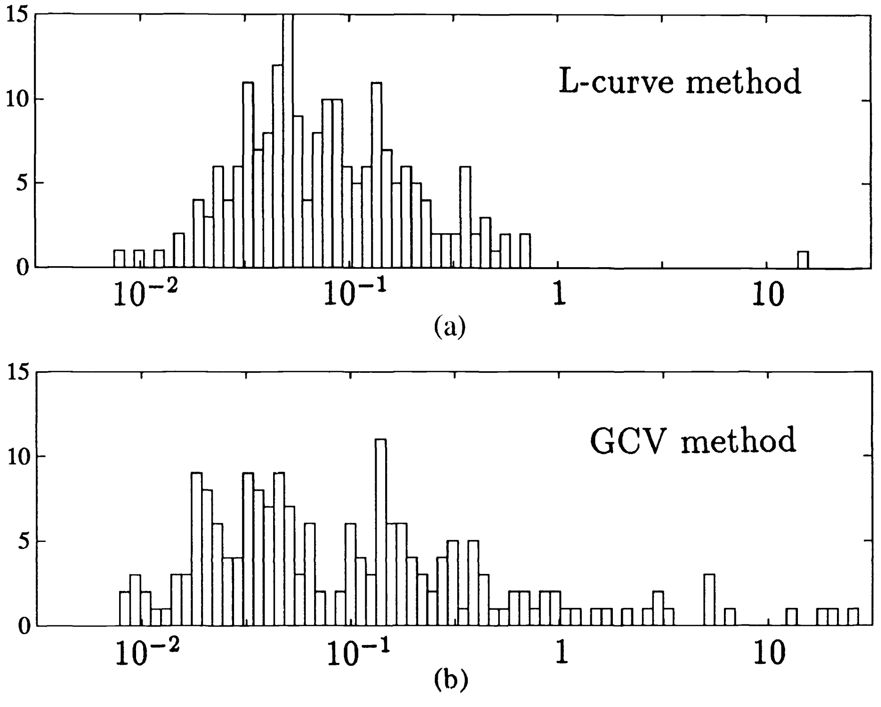
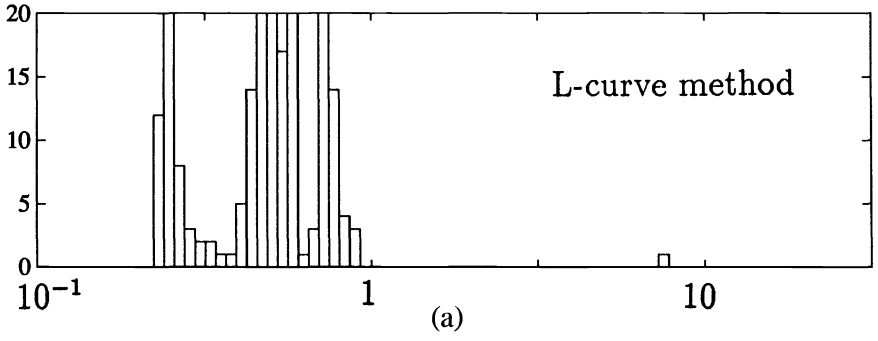


FIG. 3. *Histograms of 200 relative errors $\|x_1 - x(\lambda)\|_2/\|x_1\|_2$ for model problem one, Tikhonov regularization, and λ chosen by means of (a) the L-curve criterion and (b) the GCV. The error level $\|e\|_2/\|y\|_2$ varies between 10^{-9} and 10^{-1}.*

regularization parameter, while the GCV method fails quite often, due to the difficulties mentioned above. It is no surprise that the relative errors are typically smaller for the first model problem, because the satisfaction of the discrete Picard condition makes this problem somewhat easier to solve numerically than the second model problem.



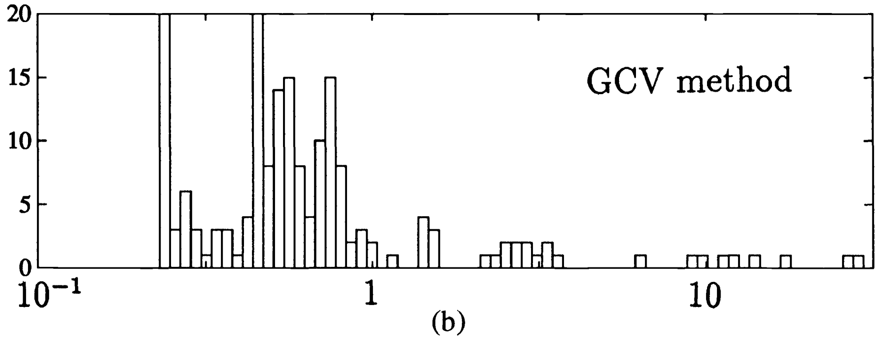


FIG. 4. *Histograms of* 200 *relative errors* $\|x_2 - x_\lambda\|_2/\|x_2\|_2$ *for model problem two, Tikhonov regularization, and* λ *chosen by means of* (a) *the L-curve criterion and* (b) *the* GCV. *The error level* $\|e\|_2/\|y\|_2$ *varies between* 10^{-9} *and* 10^{-1}.

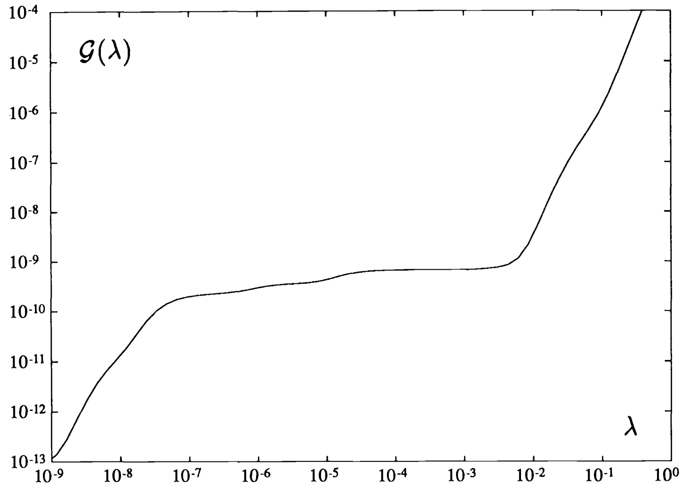
Finally, let us consider problems with highly correlated errors. For this purpose we use the first problem, but now the perturbation e is generated as follows. Once the matrix K and the right-hand-side y_1 have been computed, we smooth their elements k_{ij} and $y_{1,i}$ by the following scheme:

$$\tilde{y}_{1,i} = y_{1,i} + \mu\,(y_{1,i-1} + y_{1,i+1}), \qquad i = 2, \ldots, n-1,$$

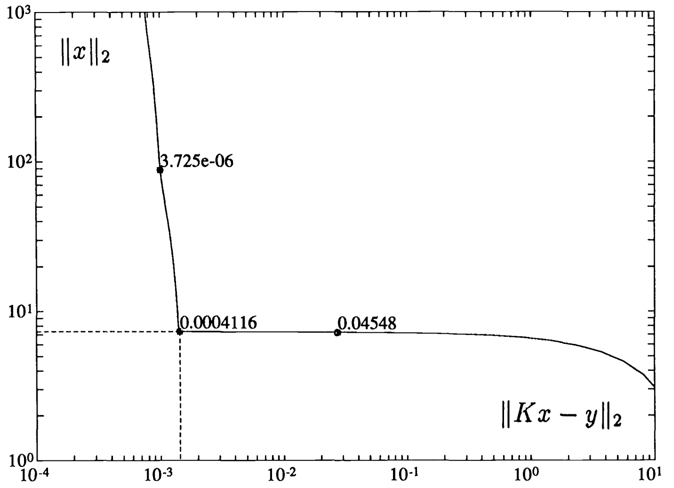
$$\tilde{k}_{ij} = k_{ij} + \mu\,(k_{i-1,j} + k_{i+1,j} + k_{i,j-1} + k_{i,j+1}), \qquad i,j = 2, \ldots, n-1.$$

Hence the right-hand-side errors are $e_i = \tilde{y}_{1,i} - y_{1,i}$, and similarly for the matrix. The parameter μ controls the amount of smoothing. These errors may, for example, represent sampling errors or the approximation errors involved in computing K and y by means of a Galerkin-type method where some "local" integration is performed. The noise is not "white" as in the first two model problems; rather, e has larger components along the singular vectors corresponding to the larger singular values.

We carried out several experiments with this third model problem for various values of μ. For all these experiments, the GCV method completely failed to compute a reasonable regularization parameter. Fig. 5 shows a typical GCV function for these experiments, for the particular choice $\mu = 0.05$. The GCV function is monotonically



(a)



(b)

FIG. 5. (a) *shows the* GCV *function for a problem with correlated errors. The* GCV *function has no minimum for any reasonable value of* λ. (b) *shows the L-curve for Tikhonov regularization for the same problem as the corner computed by our algorithm. The numbers are the regularization parameters that correspond to the dots on the L-curve.*

increasing. (In fact, there is a minimum for λ of the order of the machine precision, but this is not a useful regularization parameter.)

The L-curve, on the other hand, always has an unmistakable corner. For the same value of $\mu = 0.05$ as above, this corner occurs at $\lambda \approx 3 \cdot 10^{-4}$, and this value of the regularization parameter indeed produces a regularized solution with optimal error. There is, in fact, one more corner for λ of the order of the machine precision outside of the plot. To get the correct corner, we used $\bar{M} = 10^4$ as a huge overestimate of the solution norm. For smaller values of μ the L-curve criterion sometimes leads to an underregularized solution, essentially because, as we mentioned in §4.1, the perturbation errors must become slightly apparent in the solution to produce the corner. Nevertheless, the computed λ is still fairly close to the optimal one.

7. Conclusions. We have shown that a number of regularization methods have naturally associated L-curves defined in terms of norms that are characteristic for the particular method. We have introduced new regularization methods, based on l_p norms in the coordinate system of the singular vectors of the matrix. Moreover, we have shown that, when plotted in a log-log scale, L-curves indeed have a characteristic L-shaped appearance and that the corner corresponds to a good choice of the regularization parameter.

Based on this characterization of the L-curves, we have proposed a new a posteriori scheme for computing the regularization parameter for a given problem. This scheme uses the parameter corresponding to a corner of the L-curve, a point of maximum curvature. We have also extended this idea to discrete L-curves such as those associated with truncated SVD and iterative methods.

Our numerical examples clearly illustrate the usefulness of the L-curve criterion for choosing the regularization parameter. Although the L-curve criterion sometimes fails to compute a reasonable regularization parameter, it seems to be much more robust than its main competitor, generalized cross-validation. Of course, one can always construct problems that will also "fool" the L-curve criterion; but it is our feeling that it works so well in practice that it is indeed a useful method. Further work is needed to determine circumstances under which the regularized solution converges to the true solution as the size of the error converges to zero.

REFERENCES

[1] M. Bertero, C. De Mol, and E. R. Pike, *Applied inverse problems in optics*, in Inverse and Ill-Posed Problems, Heinz W. Engl and C. W. Groetsch, eds., Academic Press, New York, 1987, pp. 291–313.

[2] L. Eldén, *Algorithms for regularization of ill-conditioned least squares problems*, BIT, 17 (1977), pp. 134–145.

[3] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, New York, 1988.

[4] G. H. Golub, M. Heath, and G. Wahba, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215–223.

[5] C. W. Groetsch, *The Theory of Tikhonov Regularization for Fredholm Integral Equations of the First Kind*, Pitman, Boston, 1984.

[6] C. W. Groetsch and C. R. Vogel, *Asymptotic theory of filtering for linear operator equations with discrete noisy data*, Math. Comput., 49 (1987), pp. 499–506.

[7] P. C. Hansen, *The truncated SVD as a method for regularization*, BIT, 27 (1987), pp. 354–553.

[8] ———, *Regularization tools, a Matlab package for analysis of discrete regularization problems*, Tech. Report, Danish Computing Center for Research and Education, Lyngby, Denmark, 1991.

[9] ———, *Analysis of discrete ill-posed problems by means of the L-curve*, SIAM Rev., 34 (1992), pp. 561–580.

[10] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[11] F. Natterer, *The Mathematics of Computerized Tomography*, Wiley, New York, 1986.

[12] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.

[13] J. A. Scales and A. Gersztenkorn, *Robust methods in inverse theory*, Inverse Problems, 4 (1988), pp. 1071–1091.

[14] C. B. Shaw, Jr., *Improvements of the resolution of an instrument by numerical solution of an integral equation*, J. Math. Anal. Appl., 37 (1972), pp. 83–112.

[15] J. Skilling and S. F. Gull, *Algorithms and applications*, in Maximum-Entropy and Bayesian Methods in Inverse Problems, C. R. Smith and W. T. Grandy, Jr., eds., D. Reidel Pub. Co., Boston, 1985, pp. 83–132.

[16] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*, Wiley, New York, 1977.

[17] J. M. Varah, *Pitfalls in the numerical solution of linear ill-posed problems*, SIAM J. Sci. Statis. Comput., 4 (1983), pp. 164–176.

[18] G. Wahba, *Spline Models for Observational Data*, CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 59, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.